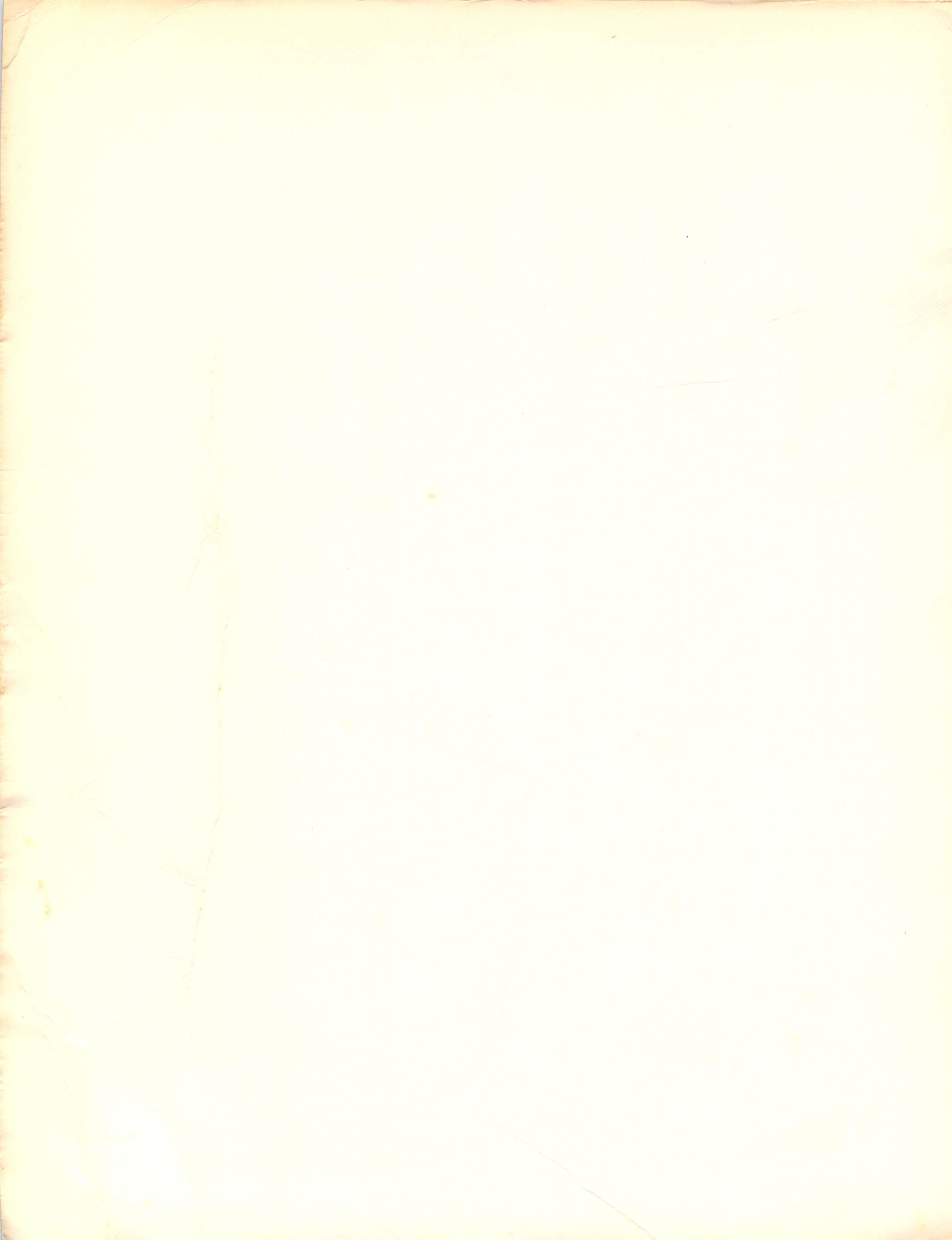
Kenneth P. Goldberg

LEARNING COMMODORE™ 64 LOGO TOGETHER

An Activity Book for Creative Parents Teachers, and Kids

MICROSOFT® PRESS

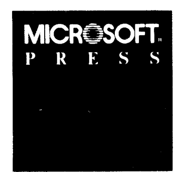


LEARNING COMMODORE 64 LOGO TOGETHER

Kenneth P. Goldberg

LEARNING COMMODORE 64 LOGO TOGETHER

An Activity Book for Creative Parents, Teachers, and Kids



Illustrated by Mits Katayama

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
10700 Northup Way, Box 97200, Bellevue, Washington 98009

Copyright © 1984 by Kenneth P. Goldberg
All rights reserved. No part of the contents of this book
may be reproduced or transmitted in any form or by any means
without the written permission of the publisher.

Library of Congress Cataloging in Publication Data
Goldberg, Kenneth P.
Learning Commodore 64 Logo together.
Includes index.
1. Commodore 64 (Computer)—Programming.
2. LOGO (Computer program language) I. Title.
II. Title: Learning Commodore sixty-four Logo together.
QA76.8.C64G65 1984 001.64'2 84-16629
ISBN 0-914845-24-1

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 FGFG 8 9 0 9 8 7 6 5 4

Distributed to the book trade in the United States and Canada
by Simon and Schuster, Inc.

Commodore™ is a trademark of Commodore Electronics Limited.
The Logo software is a copyright © of the Massachusetts Institute of Technology, 1981.

To Jeanne, Timothy, and Rebecca
for their patience and support.

Contents

Introduction ix

1. Playing with a Turtle 1

2. Getting the Turtle Running 9

3. Drawing with Turtle Graphics 23

4. The Colorful Turtle 47

5. The Turtle as Grid Hero 59

6. Writing Your Own Commands 77

7. A Closer Look at Procedures 107

8. Adding Variety with Variables 131

9. Drawing Stars and Spirals 153

10. Nonstop Procedures 187

11. Looking into Sprites 209

12. Growing and Cultivating Sprites 233

13. Working with Numbers 259

14. Advanced Computational Commands 271

15. Turning the Turtle's Pen to Prose 301

16. Activities That Speak For Themselves 327

17. Sound and Music 349

Appendix: How to Create Your Own Logo Activity Disk 369

Index 419

Introduction

In less than 10 years, personal computers have become an integral part of our society, causing a revolution in the way we live and do business. Today, another computer revolution is changing the way our children learn. One of the reasons for this revolution is a computer language called Logo.

Why Logo?

While many computer languages have been used for education, Logo is the only language developed by educators, with education as its primary goal. Logo lets children use the computer in a more natural manner, providing a controlled, but exceptionally rich environment in which children can learn about mathematics, computers, language, and the world around them through guided exploration and discovery. By providing this special environment, Logo allows children to proceed at a pace that is right for them, and to explore the parts of the Logo environment that are individually most interesting and challenging.

But Logo isn't just a child's language. While it's designed to be easy to learn, it can also be used for many different purposes and at many different levels of understanding and complexity. In other words, Logo grows with children as their own abilities grow; it is just as useful for adults and students in secondary schools and colleges as it is for students in elementary schools.

Logo is not only easy to learn, it also incorporates important programming techniques that are used in more advanced languages. Because of this, Logo is recommended by many educators as an appropriate first language for anyone who wants to learn about computers and computer programming.

How This Book Will Help You

For all these reasons, teachers all over the country are learning Logo themselves and teaching it to their students, from elementary school on up through college. In addition, many teacher-training institutions now offer workshops and courses in Logo for current and prospective teachers. If your children are learning Logo in school—and most likely they are or soon will be—this book will help you understand what your children are doing at school and what they are talking about when they come home.

Though you don't have to be a parent to read and learn from this book, the organization is tailored for the parent and child. This book shows you, the parent, what Logo is all about and how to use it, and shows you how to help your children get the maximum educational benefit from the language.

Special Features of This Book

As I just mentioned, the educational philosophy underlying Logo is that children learn best when they are allowed to explore and experiment in a controlled, but rich environment such as the one Logo provides. However, in order to experiment, they must know what they can experiment with. In a classroom, guidance is provided by the teacher. When you use Logo with your children at home, the guidance must come from you.

Of course, no one expects you to be as knowledgeable about Logo, or as skilled in guiding your children through its many features, as a teacher who has used and taught Logo for several years. After all, you're just learning Logo yourself. So, throughout the book, passages called Turtle Tips help you in your capacity as a Logo guide. These passages, marked by the turtle shown in Figure I-1, include recommendations for special activities your children can use as vehicles for exploring the different capabilities that Commodore 64 Logo offers. Turtle Tips also tell you how Logo ties in to the mathematics your children are learning in the classroom and how you can use Logo to enhance what your children learn in school.

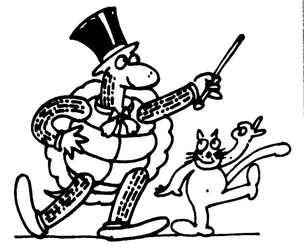


Figure I-1. The turtle marking Turtle Tips

Two other special features appear throughout this book: More Things to Do and Turtle Traps.

More Things to Do, which appear at the end of each section, are recommendations for Logo activities that help your children practice the special commands and capabilities introduced in that section. They are intended to encourage exploration and discovery, just as Logo itself was designed to do.

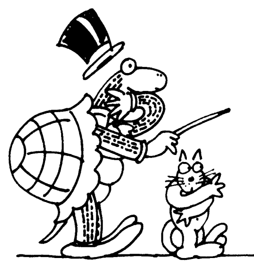


Figure I-2. The turtle marking Turtle Traps

Sometimes they also introduce new ideas that are discussed in detail later in the book. So, it's a good idea to try the activities in these sections as you proceed through the book.

Turtle Traps appear whenever a subtle or tricky point is coming up that might cause you and your children trouble unless you're forewarned. Turtle Traps are indicated by the turtle illustrated in Figure I-2.

The Logo Activity Disk

Throughout this book, Commodore 64 Logo's capabilities are illustrated and demonstrated through a variety of games and activities in Logo Activity Time sections that require use of a Logo Activity Disk. You can make your own Activity Disk using the chapter-by-chapter listings and instructions provided in the appendix. You'll need a blank disk that has been prepared to accept Commodore 64 Logo commands. You can create your Activity Disk in one of two ways: You can create the disk an activity at a time, as you come to each new chapter. Or, you can read the entire book, skipping the Logo Activity Time sections, then create the disk and go back through the Activity Time sections later.

Organization of the Book

One of the nice features of Logo is that, as you learn to use its different features, you can combine them to produce more exciting and sophisticated results. This book follows the same pattern of development. For example, the first few chapters concentrate on Logo's graphics capabilities. When you go on to use text, you'll learn how to combine text and graphics. By the time you have finished the book, you'll not only know how to use each of Logo's capabilities by itself, you'll also

know how to combine them for maximum usefulness and effectiveness.

In the same way, the programming techniques you learn and use in any one chapter will be used whenever appropriate in succeeding chapters. So, any feature or technique used in a later chapter will be one you have already learned.

How to Use This Book

The most effective way to use this book is to read over a section or two at a time, trying out the activities on the computer by yourself, before sitting down in front of the computer with your children. Once you've mastered the material and feel comfortable with it, you can go over those same sections with your children, discovering and enjoying Logo's capabilities together. Reading ahead and trying the activities beforehand allows you to plan the amount of material you can cover with your children at any one time and lets you master the technical aspects of the language on your own, so that you can concentrate on helping your children when you go over it together. Logo is a doing language rather than a reading language. So you and your children will get more out of this book if you can try out the examples and activities on your own Commodore 64 computer as you go along.

As I mentioned earlier, Logo is as useful for adults and students in secondary schools and colleges as for students in elementary schools. Because of this inherent usefulness of the language to a person of any age, there is something in almost every chapter of this book for an elementary student and there is something in almost every chapter for a high-school

student. But not every topic in every section of the book is appropriate for everyone. By reading ahead, trying the activities on your own, and using the information provided in the Turtle Tips, you can identify those topics that are appropriate for your children and those that are not.

What You Need

To follow along with the activities described in this book, you'll need:

- A Commodore 64 computer system.
- A Commodore 64 Logo package.
- Some blank disks.

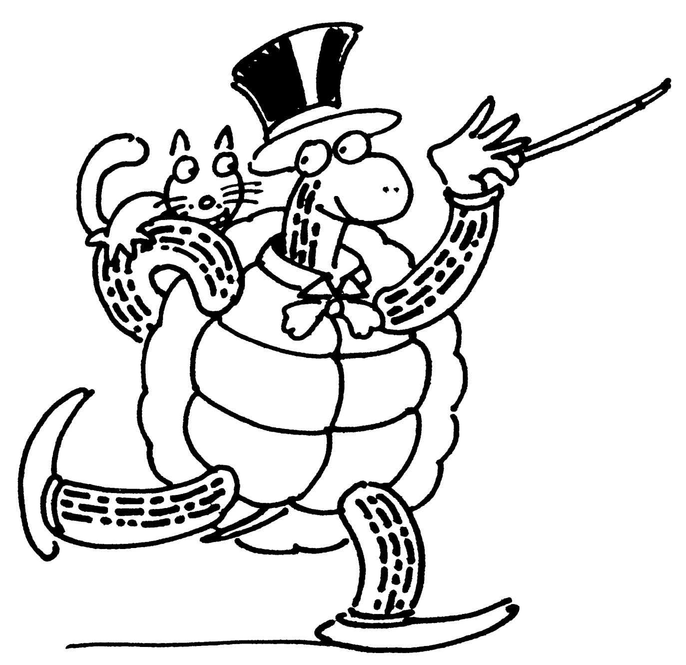
"Commodore 64 computer system" refers not just to the basic computer, which has a built-in keyboard, but to the computer, a disk drive, and a display monitor or television set. The disk drive and monitor or television set aren't normally sold with the computer; they must be purchased separately. I strongly recommend using a color monitor or television set. Logo is most effective when you can make use of its color capabilities.

The Commodore 64 Logo package includes a Logo language disk, a utilities disk, and an instruction manual. The language disk, labeled "Logo," contains the Logo language itself. The language disk enables your computer to "think" and "speak" in Logo, rather than in its "native language," BASIC. The utility disk, labeled "Logo Utilities," contains special activities and features that you can use once the language has been loaded into the computer from the language disk.

Before You Begin . . .

Finally, remember that Logo is not just a language. It's an environment in which children can proceed at their own pace and learn through the exploration of those aspects of the environment that are individually most interesting and challenging. Help your children to discover what Logo can do, but encourage them to use their imaginations in creating new ways to apply these features.

1 Playing with A Turtle



Logo is a unique language. Designed and developed specifically for education, Logo helps children explore and experience not only mathematics and computers, but the thinking process itself. That's an important goal, and one that Logo has fulfilled admirably. But Logo has turned out to be much more than just an educational tool.

Logo—especially the version that runs on the Commodore 64—is extremely easy to

learn and can be used in many ways for both work and play. Using Logo, you and your children can draw designs, compose music, write poetry, develop stories, and solve mathematics problems for school, home, or business. You can even use Logo to create your own computer games and to develop useful home applications, such as a directory of names, addresses, and telephone numbers.

Before we start exploring the many wonderful capabilities of Commodore 64 Logo, let's see how and why Logo was developed and why it has become so successful and popular, both in schools and in the home.

Gearing Up for Mathematics

Logo was developed in the 1960s by a group of computer scientists and educators at the Massachusetts Institute of Technology (MIT), working together with computer professionals at the Cambridge, Massachusetts firm of Bolt, Beranek, and Newman. This group, now commonly referred to as the MIT Logo Group, was directed by Seymour Papert, an educational visionary who sees the increasing availability of affordable computers as an opportunity to introduce children to mathematics—and mathematics to children—easily and naturally.

In his book *Mindstorms: Children, Computers and Powerful Ideas* (Basic Books, 1980), Papert recalls his childhood fascination with gears and the interaction between them as they turned. When he studied mathematics in school, he found that gears served him well as a physical model that helped him understand certain mathematical concepts. Gears provided him with an "object to think with."

Later, as a professional mathematician and educator at MIT, Papert saw computers as a means of creating another type of object to think with—one that other children could use as he had used gears. He and the MIT group set out to develop a computer language that would be easy enough for

children to learn, but one that they could use to explore mathematics, computers, and the world around them. That language was Logo; the "object to think with" was a mechanical turtle that made the language fun to learn and use.

The First Logo Turtle

The original Logo turtle was a transparent plastic "turtle shell" on top of a round base about two and a half feet in diameter that rolled around on a pair of wheels. The turtle was connected to a large computer by electrical cables, and Logo commands were used to move the turtle about the floor on a large sheet of paper. The turtle shell contained a pen that could be lowered with a command so that the turtle would leave a trail on the paper as it moved. The turtle and the Logo language that directed its actions formed what Papert called a "mathland environment."

Testing the Turtle

In the field testing that accompanied the development of Logo, Papert's hopes for the new language were overwhelmingly satisfied. Children from many geographic areas, with a wide range of talents and abilities, easily learned the basics of Logo and began to explore its many capabilities. They explored mathematical concepts such as distance, direction, motion, and color; they explored programming techniques and began to learn what computers are all about; they learned about the English language and how to use it. Logo was an enormous success with students and teachers alike.

Mathematics—Naturally

Now that we know why Logo was developed, let's see why it's been so successful.

The fact is, many people have developed a deep-seated fear of learning and using mathematics, a sort of math phobia. Papert believes that a major reason for math phobia is the unnatural way in which mathematics is traditionally taught in school. He argues that children learn to talk and get along in the world by exploring their environment in a natural way, experimenting with the things they want to do. Similarly, it is Papert's opinion that children can learn about mathematics, computers, and the thinking process if they are allowed to explore and experiment naturally and uninhibitedly in the "mathland environment" provided by Logo and the Logo turtle.

How Logo Helps

To help children (and adults) explore and develop good thinking habits and problem-solving techniques, Logo was given two special features: vocabulary extension and modular programming.

Vocabulary extension doesn't refer to words, but to commands. Logo comes with built-in "primitive" commands that you can use right away; but it also allows you to extend this command vocabulary by using the primitive commands as building blocks to define new commands of your own.

Logo's vocabulary-extension capability parallels our own learning process—going from the simple to the more complex, so expanding Logo's vocabulary encourages children to think in terms of combining simple ideas and concepts to form more complex ones.

The second special feature, modular programming, allows you to break complicated problems down into simpler modules, or components. Then, you can work on each of these components independently, and the components can be

recombined to solve the original problem. For example, individual children or groups of children can use Logo independently to draw the parts of a house—the frame, roof, door, windows, and so on. Then, the components can be combined to form the picture of the complete house.

Modular programming is a powerful technique that can be used in many other ways. Some advanced computer languages also have modular-programming capability. One such language, Pascal, is popular in colleges and is the language used in the computer science Advanced Placement Examination for high-school students going on to college. Logo thus provides a good introduction to programming for those who may want to learn other, more advanced languages later on.

Commodore 64 Logo

Almost 20 years have come and gone since the first version of Logo was developed and tested, and a number of important changes have occurred in the use of computers for education. Personal computers have been developed, and so have versions of Logo that can be used on these machines.

To make Logo easy to learn and use, yet capable of powerful and widely diverse applications, Papert and the MIT Logo group found it necessary to build into the language many commands and capabilities that other computer languages did not have. These requirements meant the original version of Logo could be used only on relatively large computers. It's only in the past few years that personal computers have been developed with enough memory to support versions of Logo. Today, however, versions of Logo are available for most popular personal computers. Commodore 64 Logo, the version you are using, is one of the newest.

Here are some of the features your Logo offers you:

■ Commodore 64 Logo's graphics features allow you and your children to create colorful pictures. If you have a

printer with graphics capability, you can even print screen pictures on paper.

■ Commodore's version of Logo also allows you to animate your drawings by using images called "sprites" that you can shape and color as you wish and then set in motion across your display screen.

■ Logo's arithmetic and mathematics features can be used for many home and business applications. And your children can use Logo with the mathematics they are learning, from elementary school on up through college.

■ The sound-generating capability of Commodore 64 Logo allows you and your family to learn about such musical concepts as pitch, duration, and rhythm. Once you learn the basics, you can develop and play your own musical compositions or add sound effects to games and other activities you create with Logo.

■ Commodore 64 Logo allows you to write and work with text. Logo's text-handling feature, called list processing, makes it easy to work with words and sentences. You can write poetry, compose letters, and create games and educational activities that mix graphics, numbers, sound, and text.

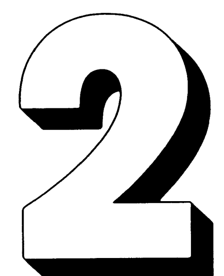
More Than Child's Play

Logo, then, is a powerful, full-fledged computer programming language. It is easy to learn and use, but it is neither superficial nor simply a children's language. In fact, as I've already mentioned, Logo incorporates important programming techniques, such as modular programming, that are typical of advanced programming languages. And, Logo is one of the few computer languages that allow programs to change their own structure while they are running. Although this very advanced use of the language isn't covered in depth in this book, it may interest you to know that this ability is a

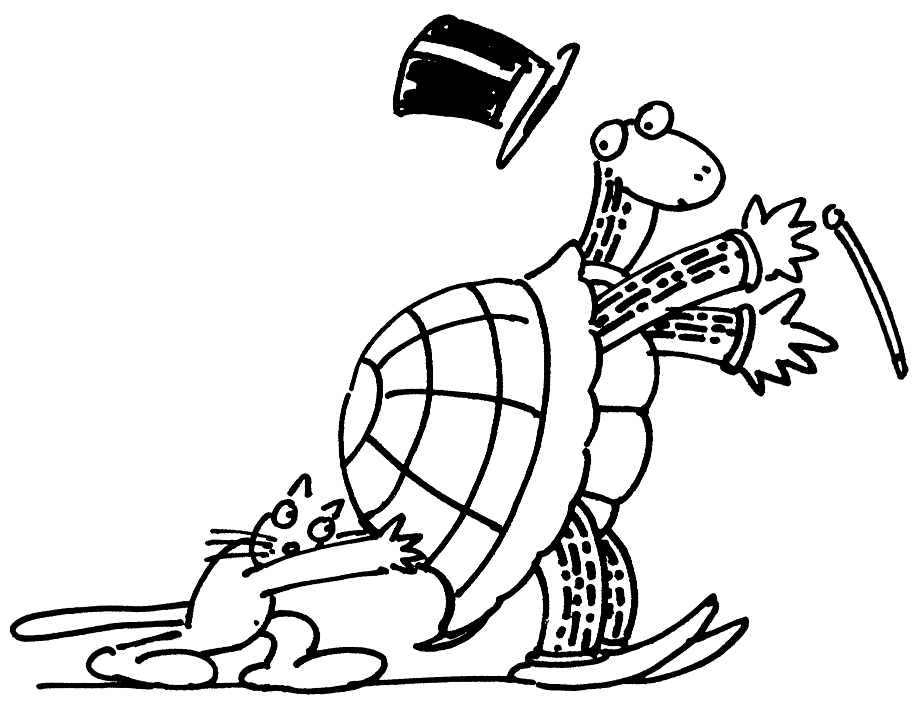
basic tool used by scientists attempting to develop "artificial intelligence"—that is, a computer language that approaches the human ability to learn from past experience and adapt to new situations.

Because Logo teaches good programming habits and because the transition from Logo to advanced languages is so easy, many educators now recommend Logo as a more appropriate first programming language for schoolchildren than BASIC, the most common beginners' language. In fact, Logo is now generally regarded as an appropriate first computer language for anyone who wants an easy-to-learn and easy-to-use language with powerful and wide-ranging applications. As you and your children progress through this book, you will learn about these important techniques and applications as you develop your programming skills.

Now that you know the what and why of Logo, you're ready to get your hands on the computer and begin to see what you can do with Commodore 64 Logo. So take out your Logo language disk, sit down at your computer, and turn the page. Your exploration is about to begin.

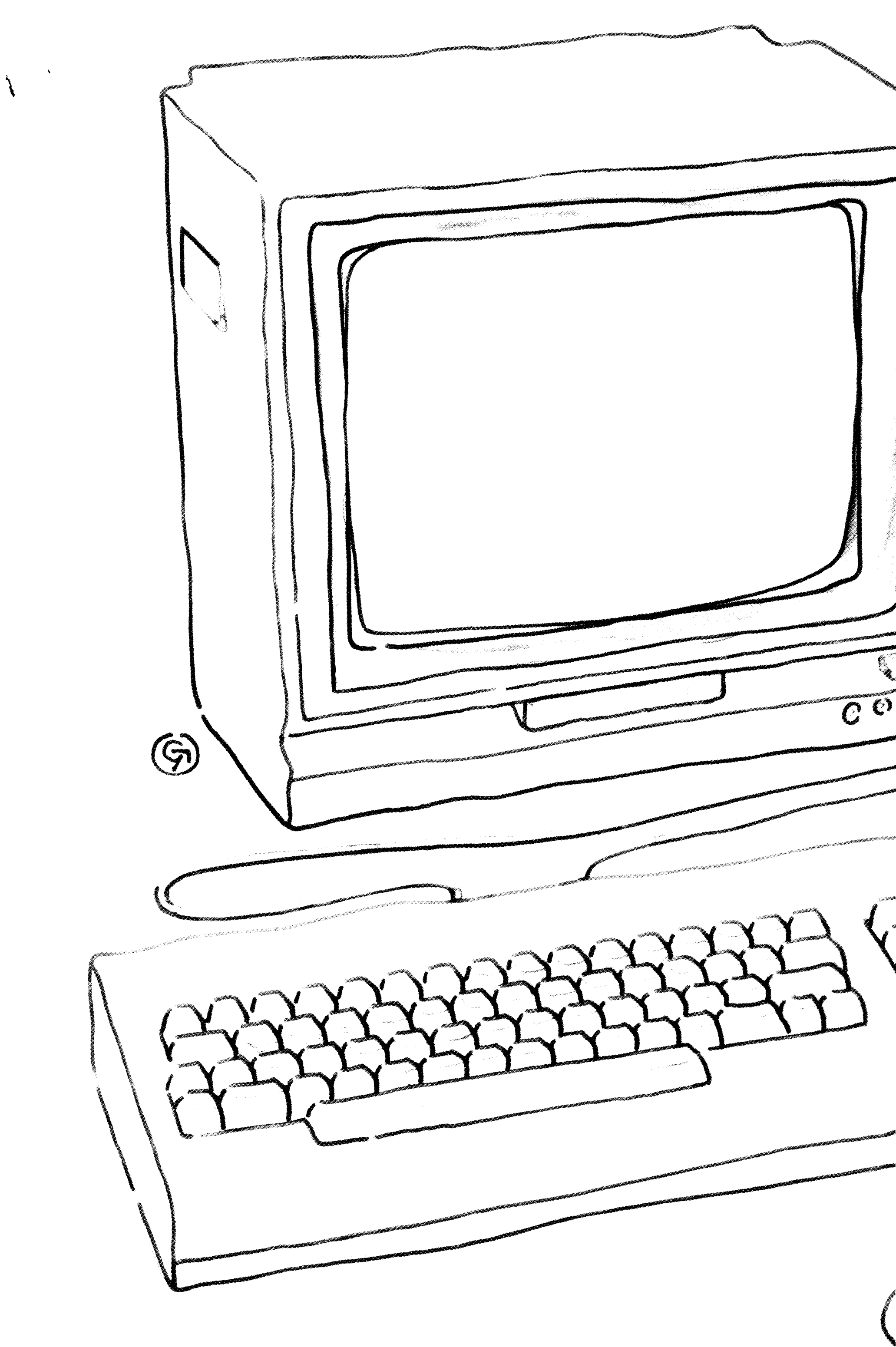


Getting the Turtle Running



If you're new to computers or to your Commodore 64, you may need a little help getting started. This chapter shows you how to turn on your computer, use the keyboard, and transfer the Logo language from the language disk to your Commodore 64. If you already know how to perform these basic operations, you may want to skip ahead to Chapter 3, where you'll start finding out how to use Logo. Now, let's get down to the basics.

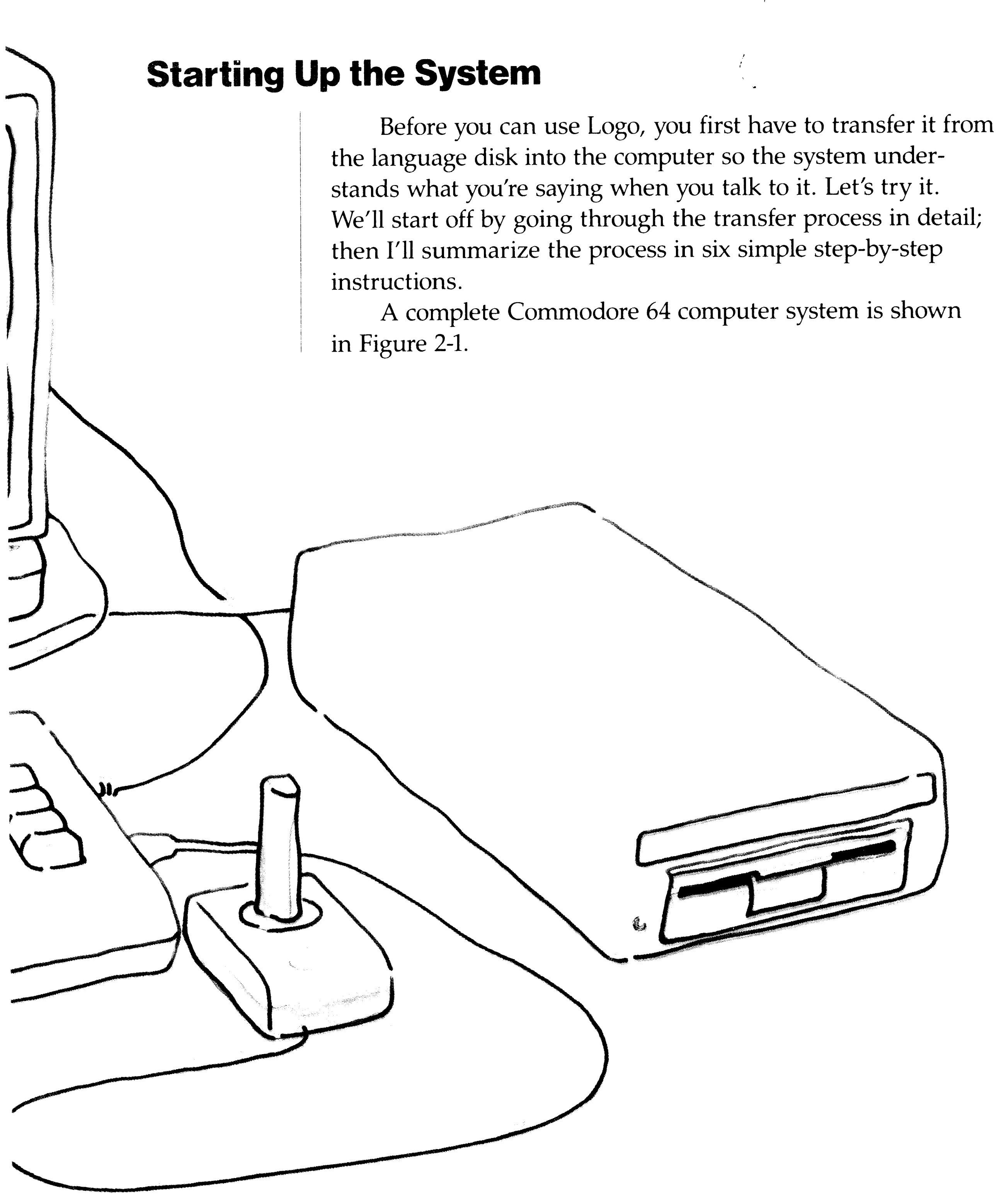
Figure 2-1.
A complete
Commodore 64
computer system



Starting Up the System

Before you can use Logo, you first have to transfer it from the language disk into the computer so the system understands what you're saying when you talk to it. Let's try it. We'll start off by going through the transfer process in detail; then I'll summarize the process in six simple step-by-step instructions.

A complete Commodore 64 computer system is shown in Figure 2-1.



To begin, first make sure your computer, disk drive, and monitor are all turned off. Then, check to see that your system is connected and plugged in. The Commodore 64 owner's manual makes a special point of telling you to turn on the computer only after you've turned on the monitor and disk drive. Similarly, the manual advises you to turn off the computer before you turn off the monitor and disk drive. The company apparently has reason to believe that turning the components of the system on or off in any other order might damage the system. Since the manual makes such a point of it, you would be well advised to do it this way.

The next step is to turn your monitor on.

Next, turn your disk drive on. The on-off switch for the disk drive is located on the back of the drive next to the cable connections. When you turn on the disk drive, the green and red lights on the front of the disk drive go on. After a few seconds, the red light goes off and only the green remains on. The green light indicates that the drive is plugged in and turned on. The red light only goes on when the disk drive is first turned on and when the drive is transferring information to or from a disk, serving as a warning that the disk drive is in the middle of some activity.

A word of caution: Don't open or close the disk drive door, or try to insert or remove a disk while the red light is on.

After you turn on the monitor and disk drive, turn on the computer. The computer's on-off switch is located on the right side next to the power plug. When the computer is turned on, a red power light near the upper right corner goes on to show that the computer is plugged in and working.

At this point, the monitor screen should look like the one in Figure 2-2.

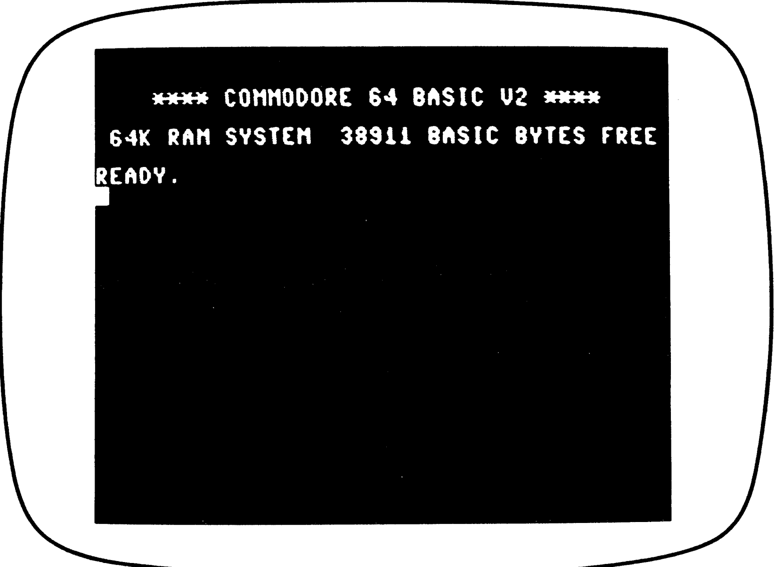


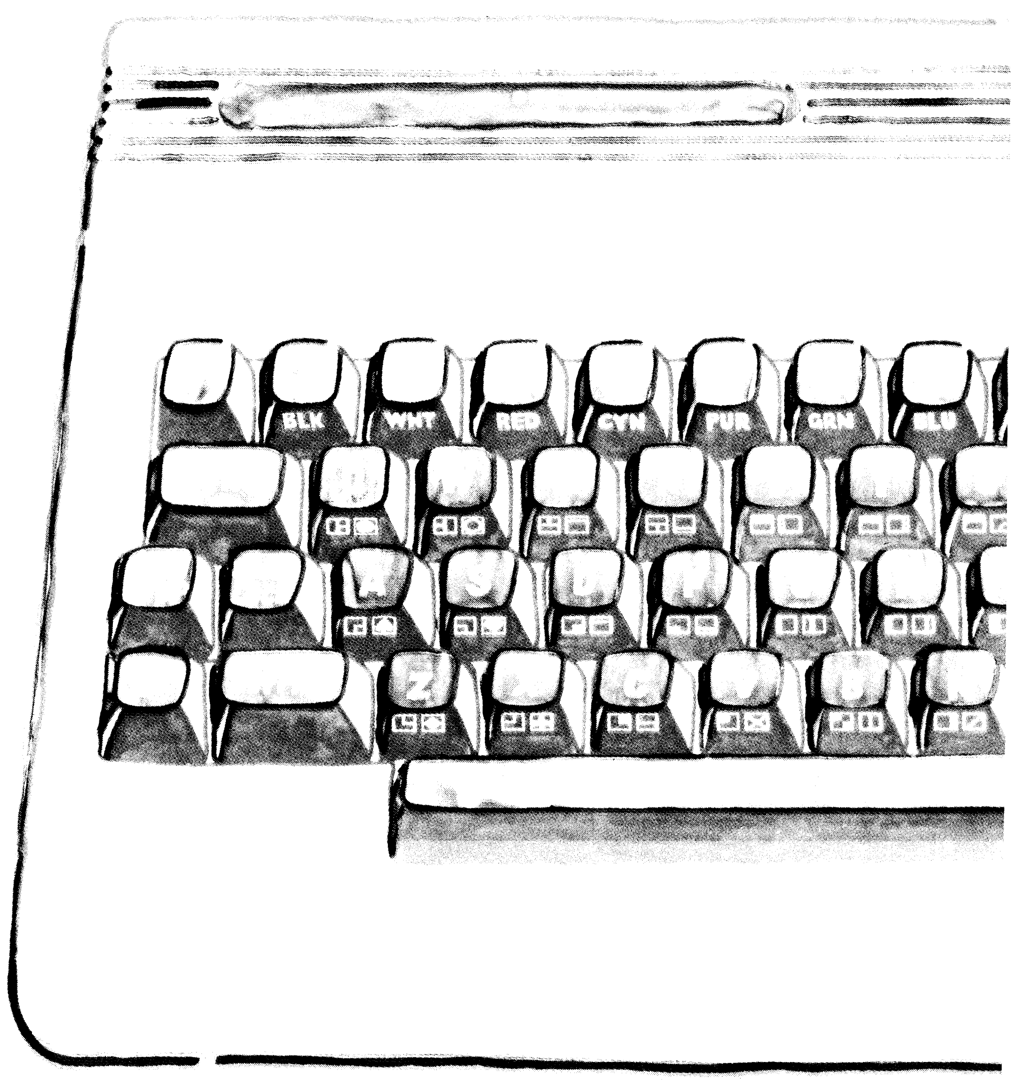
Figure 2-2. The screen of the monitor after you've turned on the computer

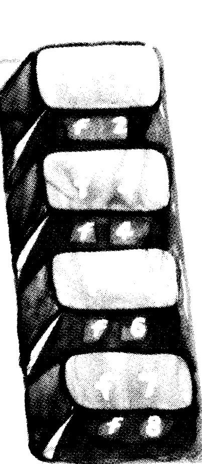
Notice the box blinking on and off just below the word READY on your screen. This is the cursor, a symbol that marks your place and shows you where the next character you type will appear on the screen.

Keyboard Conventions

In addition to letters and numbers, the Commodore 64 keyboard, shown in Figure 2-3, has certain keys with words (or parts of words) on them, such as SHIFT and DEL (short for Delete). These keys have special built-in functions that you'll learn as you go along. But it could get confusing trying to differentiate between words I'll use that refer to one particular key and words that have to be typed, since both are in all capital letters to set them off from the text. To make this distinction between keys and words, the names of keys are enclosed within the symbols < and >. So, when you see <DEL>, use the key with the word DEL on it, but when you see LOAD, type the word itself.

Figure 2-3.
A Commodore 64 keyboard





As you type, you'll notice that your screen automatically displays only capital letters. You'll need the <SHIFT> key only for typing certain characters, such as quotation marks, much as when you use a typewriter.

Loading Logo

If the door of the disk drive is closed, you can open it by simply pushing the protruding little handle in, which causes the door to spring open. Now that the system is on, you can load Logo. Take the Logo language disk (remember, it's the one that just says "Logo" on the label) out of the protective envelope. Hold the disk carefully by its label. Never touch the parts of the disk visible through the openings in the covering, since this might damage the disk. Place the disk in the long, narrow horizontal opening on the front of the disk drive with the disk's label face up and toward you, as shown in Figure 2-4. The disk should slide smoothly all the way in. Once the disk is all the way in, close the drive door by pulling the little handle down and toward you.

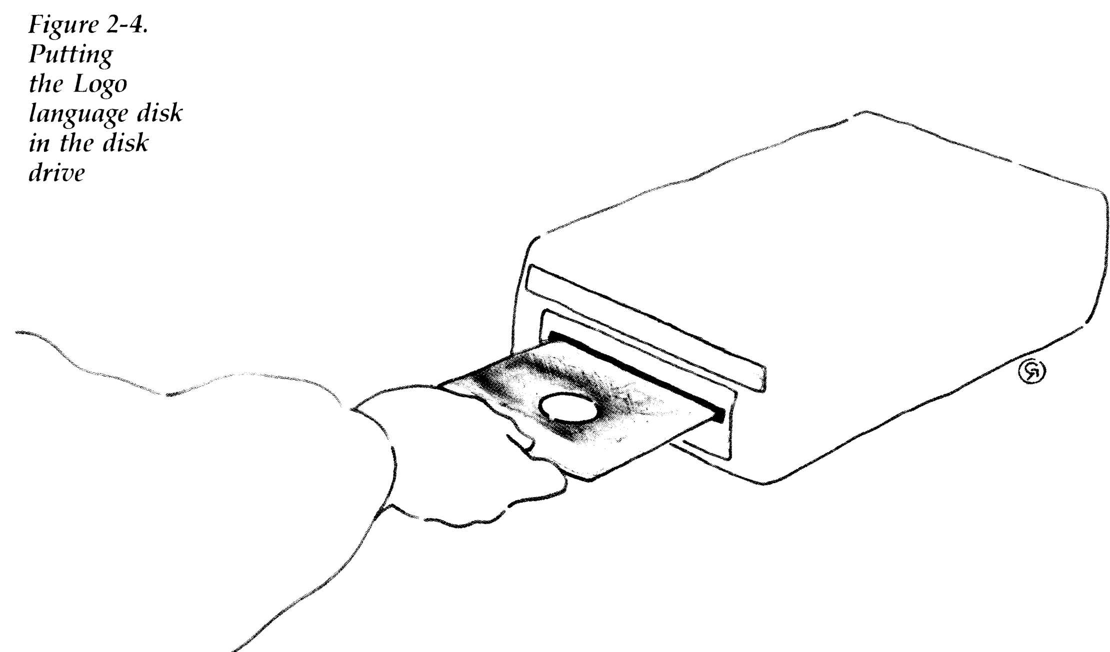
Use the keyboard to type the command:

```
LOAD "LOGO",8
```

As you type the command, the cursor moves along with you. When you've finished typing the command, press the <RETURN> key on the right side of the keyboard. Pressing the <RETURN> key tells the computer you have finished typing in your message and now you want it to read the message and do what the message says to do.

Keep in mind that the command LOAD "LOGO",8 should be typed exactly as it's shown. If you don't talk to a computer in exactly the words and phrases it's been taught to expect, it won't understand what you're saying. Unlike people, computers cannot look at a command and say to themselves, "That's not exactly what I expected. But it's close enough, so I'll accept it." If the command does not match

Figure 2-4. Putting the Logo language disk in the disk drive



exactly what the computer expects, it's as bad as if the command were typed backwards.

Let's take a look at what each part of the command LOAD "LOGO",8 means. The first part (LOAD "LOGO",) tells the computer what it is going to look for (LOGO) and what it should do when it finds it (load, or read, it into memory). Loading a language into the computer means transferring the language from a disk or cassette into the computer's memory so that the computer understands commands given in that language. The number 8 at the end of this command refers to where the computer will find what it's looking for (a disk in the disk drive.) The number 8 is necessary because a cassette recorder could also be connected to the computer at the same time as the disk drive and the computer

needs to be told which of these information devices to go to for the desired information. Try typing this command with the number 7 replacing the number 8. The computer displays an error message on the screen, like the one shown in Figure 2-5, saying it looked for Logo and couldn't find an appropriate information device at location 7.

Figure 2-5. An error message appears on the screen when you type the wrong number in the LOAD command.

If you make a mistake typing a command, as soon as you press <RETURN> the computer will display an error message on the screen telling you that it doesn't understand what you're saying or it cannot do what you want it to do. If the computer responds to one of your commands with an error message, simply try typing the command again.

If you catch your mistake before pressing <RETURN>, there is a simple way to make corrections. One advantage of typing on a computer keyboard over typing on a typewriter is that you can easily correct any mistakes you make. You'll learn several ways of making corrections, but one way is with the delete key <DEL> at the upper right corner of the keyboard. (This key also has the letters INST printed on it just above

DEL. INST stands for insert, which is a secondary function of the key that you activate if you press it together with the <SHIFT> key. For now, you'll use this key only for deleting, so you needn't worry about INST.) If you make a mistake or if you just decide to erase what you have typed, press the <DEL> key to move the cursor one space to the left and erase whatever character was there. The cursor continues to move left and erase characters as long as you hold the <DEL> key down. Once you've erased what was incorrect, you can type the correct characters.

If you've typed the command correctly, when you press <RETURN>, the red light on the disk drive goes on, indicating the computer is looking for information under the name Logo on the disk in the disk drive. While the red light is on, the messages SEARCHING FOR LOGO, LOADING, and READY appear on the screen, one after the other, very quickly. After about five seconds, the red light goes off and the screen shows:

SEARCHING FOR LOGO
LOADING
READY.

These messages tell you that the computer looked for Logo on the disk, found it, and is ready to make it available for your use.

Now type the word RUN and press <RETURN>. The red light on the disk drive goes on again and the screen displays the message:

Loading, please wait...

After approximately a minute and a half, LOADING, PLEASE WAIT... is replaced by the message OK in white lowercase letters. Then the word RUN in dark lowercase

letters replaces OK. When the red light on the disk drive goes off, your screen looks as illustrated in Figure 2-6 and Logo is in the computer and ready for you to use.

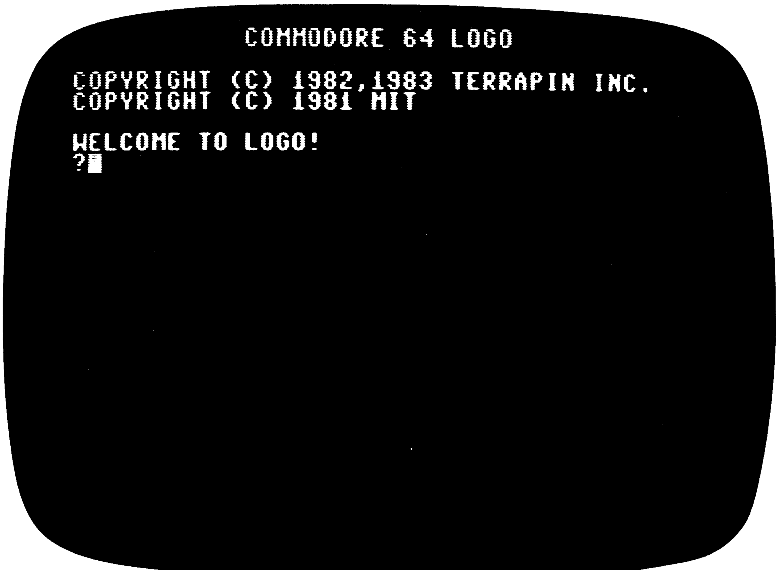


Figure 2-6. The monitor screen after you've loaded Logo

A Final Word

You can now open the disk drive door, take the language disk out, and put the disk away in a safe place. Your computer now understands the Commodore 64 Logo language and will retain knowledge of Logo until it is turned off.

The next time you turn your computer on, you can get back into Logo by following the steps just described. To help you, here are the six steps for starting the system and entering Logo:

1. Turn the monitor on.

2. Turn the disk drive on.

3. Turn the computer on.

4. Place the Logo language disk in the disk drive and close the drive door.

5. Type LOAD "LOGO",8 and press <RETURN>.

6. Type RUN and press <RETURN>.

Drawing with Turtle Graphics

Now that you know how to start up the system and load the Logo language, you will want to begin making use of Logo's graphics capabilities. To help you understand Logo better, let's explore Logo's two states, or modes: graphics and non-graphics, also known as Draw and NoDraw.

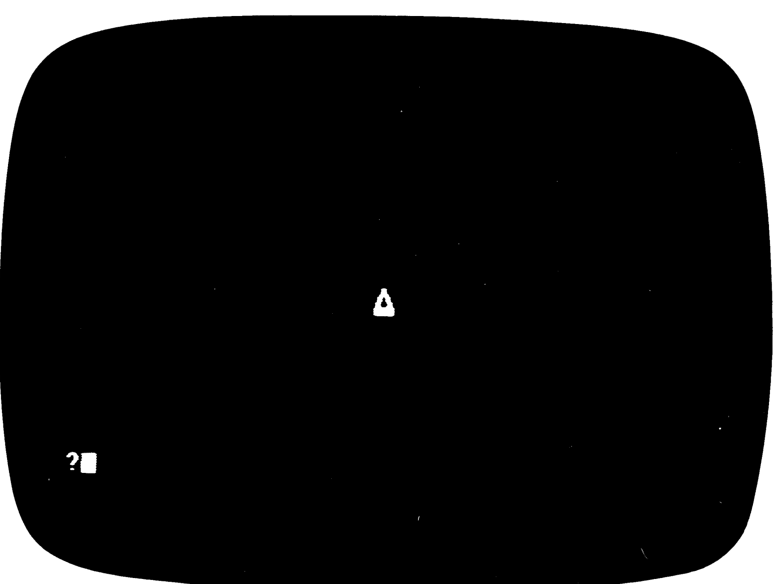
A Screen for Drawing

If you're using a color monitor, your screen should be a solid sky blue. The solid blue screen indicates you are in the NoDraw mode. In the NoDraw mode, you can write words and sentences or do arithmetic computation. You'll learn more about writing and computing later, but right now you're going to start drawing pictures on the screen. To do that, you'll need to switch to the Draw mode.

Type DRAW and press <RETURN>. (I'm going to stop saying "press <RETURN>" each time I tell you to type a command, though I'll continue to include <RETURN> periodically as a reminder. Just remember that until you press <RETURN>, the computer doesn't know you have finished your command and won't read it and carry it out. So, if you've typed your command and nothing is happening, it's probably because you forgot to press <RETURN>.)

Your screen now looks like the screen in Figure 3-1. It should have a narrow blue band across the bottom and a large gray area filling the rest of the screen. This is the Draw mode. (If the colors don't look as described, you'll want to adjust the color on your monitor.)

Figure 3-1. The screen when Logo is in the Draw mode



The Commodore 64 Turtle

The white triangular object in the center of the gray area is the turtle. Although the triangle on your screen may not look like a turtle, it's the Commodore 64 equivalent of the mechanical turtle used by the MIT Logo Group—it's an object to think with. You can use it to draw pictures and create designs, the same way the children at MIT used the mechanical turtle.

If you look closely at the turtle on your screen, you'll see it has three sides, and that the base, or bottom side, is somewhat thicker and shorter than its other two sides. The base is thicker to help you tell the turtle's front from its back and to help you tell in which direction it's pointing: Its "head" is the point opposite the thicker and shorter side. You might think of the turtle as an arrow pointing in the direction you want the turtle to move.

The large gray area containing the turtle is the drawing area where your pictures and designs will appear. The small blue area at the bottom is the text area where the commands you type will appear.

Switching Between Draw and NoDraw

Just for practice, try switching back and forth between the NoDraw and Draw modes several times. You can switch to the NoDraw mode by typing NODRAW, or ND for short; and you can switch back to the Draw mode by typing DRAW. DRAW and NODRAW are two of the many built-in Logo commands, called primitive commands or just primitives. You don't need to type DRAW when you give a drawing command while you're in the NoDraw mode; the computer automatically switches to the Draw mode for you. However, when I'm going to use Logo graphics, I generally prefer to put the screen into the Draw mode myself so I can see the screen as I begin my drawings.

For now, make sure you're in the Draw mode so you can begin moving the turtle around the screen.

The Turtle's First Steps

Four primitive commands move the turtle around the screen: FORWARD, BACKWARD, RIGHT, and LEFT. If you wish, you can abbreviate these commands as FD, BK, RT, and LT. Each of these commands must be followed by a space and a number—positive, negative, zero, or even a fraction or a decimal.

A number after RIGHT or LEFT tells the turtle how many degrees to turn in the direction you specify. Since there are 360 degrees in a circle, RIGHT 90, for example, would move the turtle a quarter turn to the right, RIGHT 180 would turn it upside down, and RIGHT 360 would turn the turtle completely around, leaving it pointing the same way it started. (The turtle turns so fast, though, you can't see it move.) A number after FORWARD or BACKWARD tells the turtle how far to move in steps. The number you type represents the number of steps you want the turtle to take. Turtle steps are very small, so to see what they are like, move the turtle forward 20 steps by typing:

```
FORWARD 20
```

or, for short:

```
FD 20
```

The turtle moves a short distance forward, leaving a white trail behind it as it goes. As I mentioned earlier, when the turtle was a mechanical object with an actual pen, it left a trail of ink on the sheet of paper over which it traveled. Now, of course, the triangular screen turtle simply leaves a trail of color over part of the screen. If it didn't move or leave a trail, check what you typed to see if you made an error. For instance, Logo expects a space between FORWARD and 20. If you didn't leave a space, Logo gave you an error message.

Twenty steps wasn't very far, so let's try a larger number of steps. To start over, type DRAW. This causes the turtle to return to its Home position at the center of the screen with

its head pointing straight up and erases any picture that was on the screen. Now, let's move the turtle forward 50 steps by typing:

```
FORWARD 50
```

or, in the short form:

```
FD 50
```

Ah, that's more like it, as you can see in Figure 3-2.

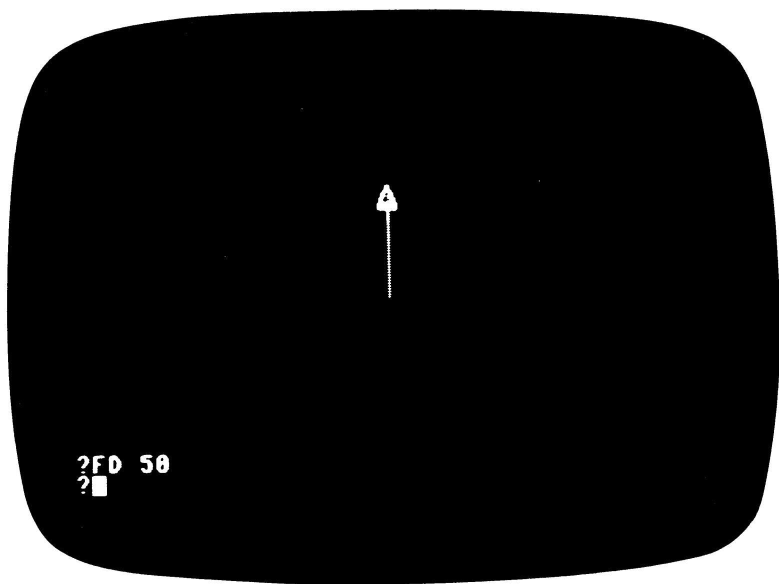


Figure 3-2. The turtle after taking 50 steps forward

Now, make the turtle turn right 90 degrees, by typing:

```
RIGHT 90
```

or, for short:

```
RT 90
```

The turtle is now pointing to the right side of the screen, as shown in Figure 3-3.

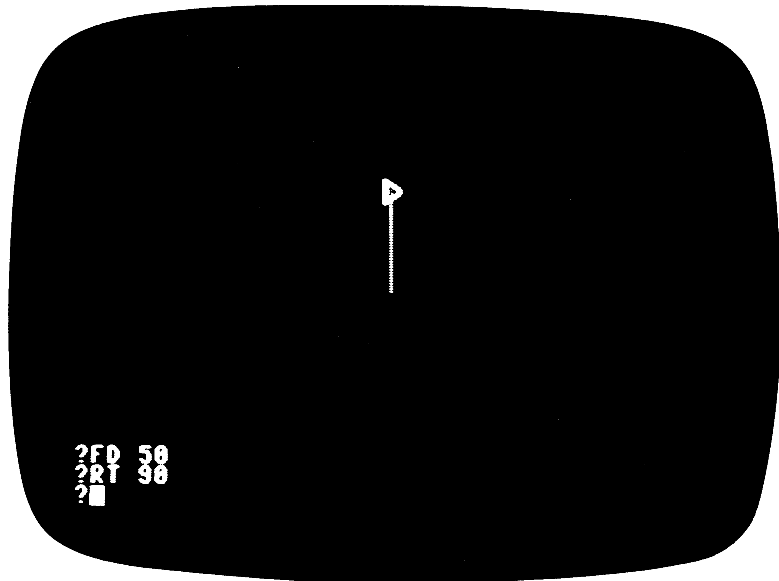


Figure 3-3. The turtle turned 90 degrees to the right

By typing these two commands three more times, make the turtle draw a square with sides 50 steps long. If you do it correctly, your picture will look like the square in Figure 3-4.

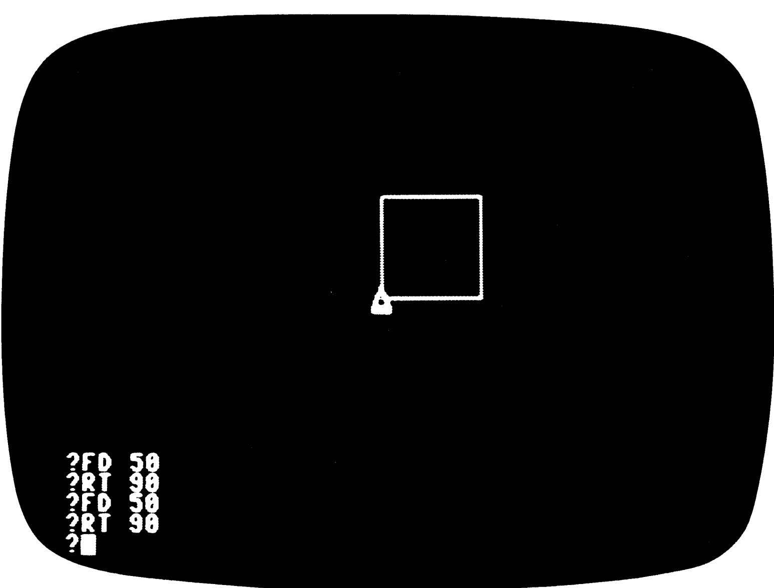


Figure 3-4. A square drawn by the turtle

Turtle Tip: It's important to remember that Logo is an exploratory language and that making mistakes is a natural and unavoidable part of exploring and experimenting. Because of this, you shouldn't give your children the feeling that they must do everything "correctly." In fact, in Logo there is no "correct" or "incorrect." There's simply the opportunity to try things and learn by experience. What is important is the encouragement and guidance you give your children, including helpful hints on how to proceed when they get stuck.

For example, suppose your child accidentally typed RT 50 instead of RT 90 while trying to draw the square. You could show your child how to simply start over again by typing DRAW and clearing the screen. But you could also show your child how turning the turtle an additional 40 degrees using RT 40 would obtain the same result as turning the turtle 90 degrees with the one command RT 90 (50 degrees plus 40 degrees equals 90 degrees).

This second approach has several educational advantages. First, it teaches your child to stop and think when something unexpected happens, instead of automatically discarding what's occurred and starting over again. Second, it gives your child practice in analyzing what happened, why it happened, and what the alternatives are. Third, it provides your child with the opportunity to think about and apply mathematical concepts in a real problem-solving situation. Mathematics becomes more meaningful and more useful when your child can apply mathematical concepts to a real situation to solve a problem he or she wants to solve.

Don't worry if you type the commands incorrectly or if the turtle doesn't do what you wanted it to do. You can always correct your typing using <DEL> if you haven't pressed <RETURN> yet. Or, you can just type DRAW to erase the picture (but leave the commands) and start over again.

A Repeat Performance

When you drew the square, you had to repeat both the FORWARD 50 and RIGHT 90 commands four times. If the space at the bottom of the screen weren't limited to five lines, the repetition of commands would look as follows:

```
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
```

You could also have typed all eight commands as one long command, as long as you put a space between each of them so the computer could see they were individual instructions. Using the abbreviations FD and RT instead of FORWARD and RIGHT to save typing and space, the single command looks as follows:

```
FD 50 RT 90 FD 50 RT 90 FD 50 RT 90 FD 50 RT 90
```

Repeating Commands

Fortunately, Logo provides a primitive, or built-in, command that allows you to perform repetitious commands much more succinctly and easily: the REPEAT command. Using REPEAT, you can replace either of the two previous sets of commands with a much shorter command:

```
REPEAT 4 [FD 50 RT 90]
```

To use a REPEAT command, type the word REPEAT, then a space, then the number of repetitions desired, then another space, then an open bracket, then the sequence of commands you want repeated, then a close bracket. (To get a left, or open, bracket, hold down <SHIFT> and tap the colon

key. To get a right, or close, bracket, hold down <SHIFT> and tap the semi-colon key. The colon and semi-colon keys are in the same row on the keyboard as <RETURN>.

Type this REPEAT command to see that it produces the same square as the list of commands used before.

Let's use this shorthand REPEAT command to make the turtle draw a triangle with sides 60 steps long.

When you drew the square, you had to make each of the four turns 90 degrees, because one complete revolution is 360 degrees. Since a square has four sides and therefore the turtle has to make four turns to complete the figure, each of the turns must be 90 degrees for the total to be 4 times 90, or 360 ($4 \times 90 = 360$). Now, you want to draw a triangle. If all the sides are to be the same length, the turtle has to make three turns of 120 degrees in order for the total of all three turns to be 360 degrees ($3 \times 120 = 360$).

To see if this reasoning actually works, type the command:

```
REPEAT 3 [FD 60 RT 120]
```

You should get the triangle pictured in Figure 3-5.

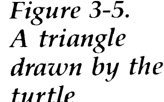
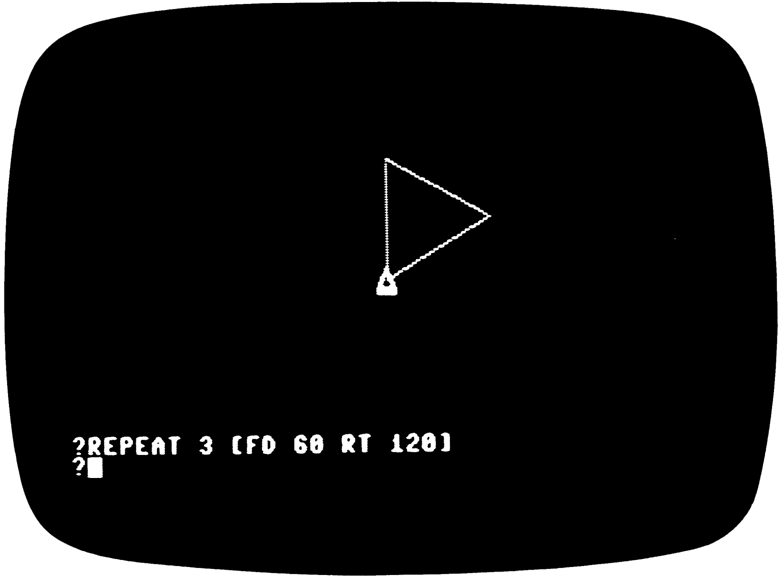
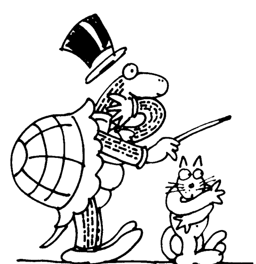


Figure 3-5. A triangle drawn by the turtle





Turtle Trap: If you try moving the turtle too far forward or backward, or if you try drawing a square or triangle that is too large for the screen, you'll notice something funny happening. When the turtle leaves the display screen, it immediately reappears on the opposite side. That is, if it disappears off the top of the screen, it reappears at the bottom; if it disappears off the right side, it reappears at the left side; and so on. It does that because the screen is automatically in what's called the Wrap mode when you type DRAW. In the Wrap mode, the screen acts as if the top and bottom edges are connected to each other and the left and right edges are connected to each other, rather like the wrapping on a birthday gift.

If you want to keep the turtle from going off the screen and reappearing at the opposite edge, you can put the screen in the NoWrap mode by typing NOWRAP. Anytime you give the turtle a command in the NoWrap mode that would cause it to leave the rectangular display screen, the computer displays the message:

```
TURTLE OUT OF BOUNDS
```

and the turtle isn't allowed to obey the command. You can put the screen back into the Wrap mode by typing WRAP or by switching to the NoDraw mode and then back to the Draw mode again.

More Things to Do

Use the REPEAT primitive command to draw figures with five sides, six sides, and ten sides. To draw these figures you have to determine, for each figure, how many degrees each turn should be, so that all the turns in that figure total 360 degrees. Based on how these three figures look, try to guess what a figure with 360 sides looks like.

Just for variety, you might want to use sides of different lengths in the different figures. If you pick too long a side for a particular figure, however, the turtle goes off the edge of the screen and reappears at the opposite edge, giving a strange-looking picture. Because of this, you may want to put your screen in the NoWrap mode first, so you automatically get a warning message whenever the turtle is going to go off the screen.

Turtle Tip: As you and your children try these activities and any others that come to mind, notice (but don't say anything to them about it) how painlessly they are learning about geometric concepts involving distances, angles, and the relationships between them. For instance, they are learning that one complete revolution contains 360 degrees, and they're also learning how to compute the number of degrees in each turn to draw a geometric figure with a specified number of sides. While this learning and discovery is enjoyable and important in its own right, it also will pay rich dividends in the mathematics classroom.

For the figure with 360 sides—a circle—try using sides one step long to start with. Then, you can decide if you want to make it larger or smaller by changing the length of the sides. As I mentioned earlier, the commands FORWARD, BACKWARD, RIGHT, and LEFT can be used with fractions and decimals as well as with whole numbers. This choice should allow you to experiment freely with creating geometric figures of all different sizes.

As you experiment with moving the turtle around the screen and drawing geometric figures, remember: There's no need for you to follow what's in this book step-by-step without deviation. If an interesting idea occurs to you, go ahead and try it out. You learn by doing in Logo, and not everyone has the same interests or ideas. So, let your imagination have free reign. You can always come back to your place in the book after you've tried out your own ideas and followed your own imagination.

Repeating Commands from the Keyboard

In the previous section, you saw how to simplify repetitious commands using the REPEAT primitive. Another way to simplify repetitious commands is by pressing a special key on the Commodore 64 keyboard. This key is just above

<RETURN> and looks like an arrow pointing straight up <↑>. If you've just typed in a command, pressing the <↑> key causes this last command to be printed again (repeated) on the following text line when you press <RETURN>. Let's see how to use the <↑> key to get some simple, but effective drawings.

Start with the command for a square, but this time, turn the turtle 90 degrees at the end so it's positioned to draw a new square when you repeat the command. The command should look as follows:

```
REPEAT 4 [FD 50 RT 90] RT 90
```

Press <RETURN> and you should get the same old square, but with the turtle pointing to the right. If you now press <↑>, the command is automatically repeated on the next text line. Pressing <RETURN> again causes the turtle to draw a second square, as shown in Figure 3-6. Use the <↑> and <RETURN> process two more times to obtain the picture of four squares shown in Figure 3-7.

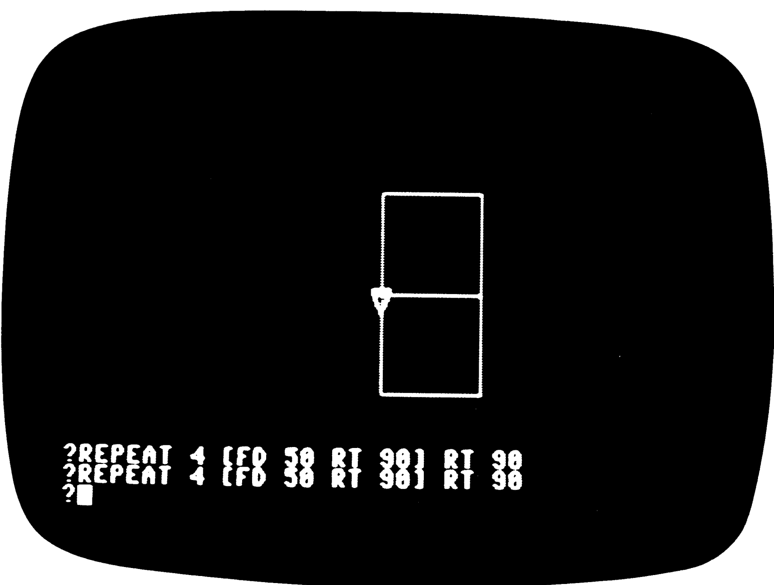
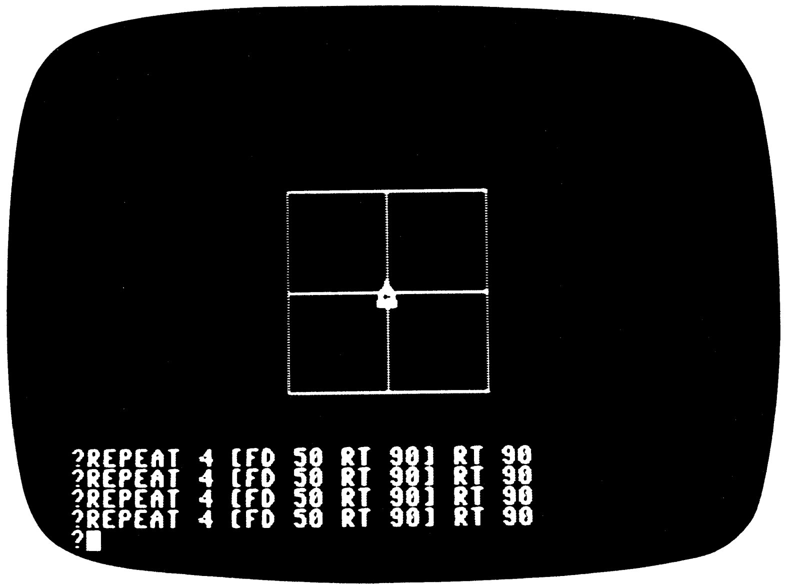


Figure 3-6. A second square drawn by pressing <↑> and <RETURN>

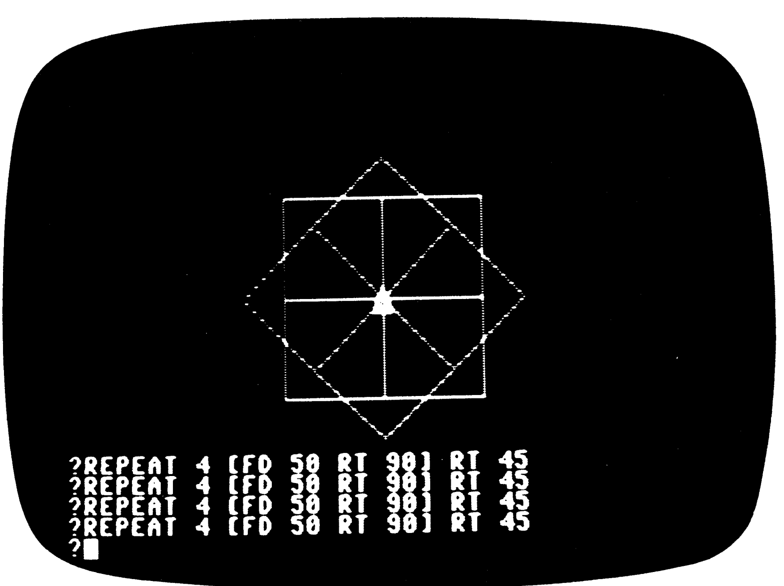
Figure 3-7. Four squares drawn by pressing <↑> and <RETURN> three times



Instead of turning the turtle 90 degrees at the end of drawing the square, if you turn it only 45 degrees, you can use the <↑> key to draw eight repetitions of the square and end with the fancier design shown in Figure 3-8. Try this yourself. The basic command to be repeated eight times is:

```
REPEAT 4 [FD 50 RT 90] RT 45
```

Figure 3-8. Eight squares turned 45 degrees



More Things to Do

Use the automatic command-repetition key <↑> and the commands for drawing a triangle to create designs with six triangles similar to the one you've created here with four squares. If you turn the turtle 60 degrees after each triangle, you get the design pictured in Figure 3-9. Turning it 30 degrees each time should result in the design pictured in Figure 3-10. You might want to try this same technique with the five-sided, six-sided, and ten-sided figures developed in the previous More Things to Do section. Or, you might want to use squares and triangles to create designs in some completely different way of your own choosing. Experiment as freely as you wish. If you don't like what you get, you can simply erase it by typing DRAW.

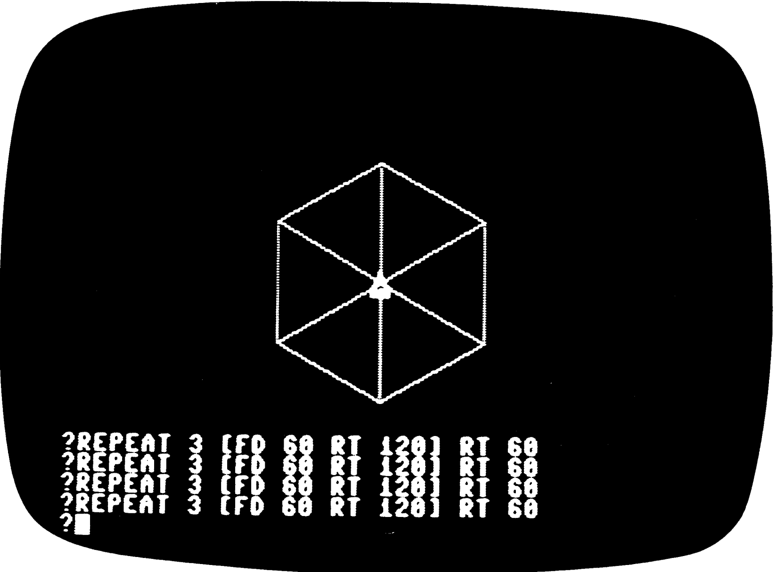
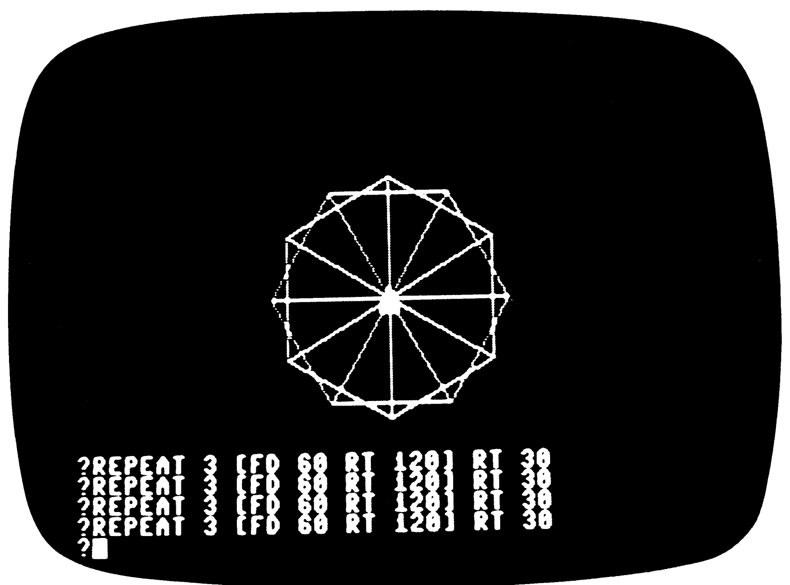


Figure 3-9. Six triangles turned 60 degrees

Figure 3-10. Twelve triangles turned 30 degrees



Leaving No Trail

Up to now you've started your drawings at the center of the screen because that's the Home position to which the turtle returns when you type DRAW. There are times, however, when you might prefer to begin your drawings elsewhere on the screen or when you might like to place several drawings at different places on the screen, without leaving a turtle trail from the Home position. Logo allows you to move the turtle to any beginning position you want on the screen, without leaving a trail, with the primitive command PENUP (abbreviated PU).

The PENUP (or PU) command in the Draw mode causes the turtle to "lift" its drawing pen so it doesn't leave a trail as it moves across the screen. The PENUP command originated in the early days of Logo when the turtle was a mechanical object with an actual pen. After you give the PENUP command, you can move the turtle to any position on the screen by using the commands FORWARD, BACKWARD, RIGHT, and LEFT. But, the turtle won't leave a trail or a line to mark its progress.

To make the turtle leave a trail again, use the command PENDOWN (or PD). Whenever you type DRAW, the turtle automatically returns to its Home position at the center of the screen with its pen down and ready to draw.

Erasing the Turtle's Tracks

Another way of positioning the turtle at a desired location, ready to draw on a blank screen, is with the command CLEARSCREEN (or CS). CLEARSCREEN clears the display screen while leaving the turtle where it is. If you move the turtle to the desired location, but accidentally draw a trail as you move it there, type:

```
CLEARSCREEN
```

and press <RETURN> or:

```
CS
```

and press <RETURN> to make the trail disappear.

Controlling the Screen Display

You may have noticed while drawing the shapes, that if a figure extends too low toward the bottom of the screen, part of it is hidden by the five lines of text in the blue text area. When you're in the Draw mode, Logo lets you use three keys on the Commodore 64 keyboard to switch between three different screen displays, called screen formats: SplitScreen, FullScreen, and TextScreen. They're all useful formats and it's nice to have all three available for those times you want to make use of them.

SplitScreen is the format you've been using while you've been in the Draw mode, the one you automatically get whenever you type DRAW. SplitScreen refers to the fact that the screen is split into an upper part where drawings can appear and a lower part where up to five lines of text can appear.

The SplitScreen format allows you to develop a picture and see what you're typing at the same time.

FullScreen allows the entire screen to display drawings, with no text showing whatever. The FullScreen format is useful if you want the screen to display a picture or design that is so large that part of it would be hidden by the text area of the screen at the bottom; or if you don't want the text at the bottom of the screen to interfere with the effect of the design.

TextScreen allows the entire screen to display text, with no drawings showing. The TextScreen format is useful if you want to see more than the previous five lines you've typed.

Switching from one of these three formats to another does not erase either text or drawings; it just changes which of these you can see at any one time. Remember, though, that when you type DRAW you will automatically be put in SplitScreen format with all prior drawings and text erased.

The three keys you use to switch between these formats when you are in the Draw mode are the light brown keys on the far right of the keyboard, labeled <f1>, <f3>, and <f5>. The <f3> key puts you in SplitScreen format, <f5> puts you in FullScreen format, and <f1> puts you in TextScreen format.

You can also switch to one of these formats with the commands SPLITSCREEN, FULLSCREEN, and TEXTSCREEN.

Since you have both graphics and text on your screen now, try using these keys and commands to switch from one format to another and see what the screen looks like in each of them.

Clearing Text on the Graphics Screen

You probably noticed that the commands you typed appeared at the bottom of the screen. Having the commands on the screen lets you see what you are typing so you can catch any errors and so you can look back to see your last few commands. When you switch to the NoDraw mode and then

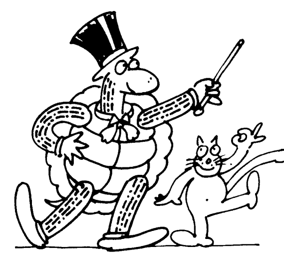
back to the Draw mode, all of your commands are erased and the bottom of the screen is blank once again. Try switching from Draw to NoDraw and back again just to see. You also can clear the text from the bottom of the graphics screen without having to switch to the NoDraw mode and back again to Draw, by using the command CLEARTEXT or by holding down the <SHIFT> key while pressing the <CLR> key near the upper right corner of the keyboard. (<CLR> also has the word HOME on it.) The advantage of clearing the text using the CLEARTEXT command or the <CLR> key instead of NODRAW followed by DRAW is that CLEARTEXT and <CLR> erase the text, but leave any picture you may have drawn on the screen.

More Things to Do

Use the PENUP command and move the turtle to each of the four corners of the screen. Draw a different figure in each corner—say, a square in the first corner, a triangle in the second, a rectangle in the third, and a circle in the fourth.

As you may recall from an earlier More Things to Do section, you can draw a circle by making the turtle go forward one step and take a one degree right turn, repeating this step-turn process 360 times. Of course, this might result in a rather large circle for only a corner of the screen. But recall that the turtle can go fractional steps forward, as well as whole steps. So, try experimenting with the size of the step, perhaps using FD 0.5 instead of FD 1. Try some different numbers of steps to see what looks best.

You might also try to use the turtle to write your initials or your name or today's date. Use PENUP and position the turtle where you want the next letter or number to begin. Then use PENDOWN to get it ready to draw. Or, first position it, and then use CLEARSCREEN to erase the trail it left getting there. If you're getting really good at this, you might want to try using the turtle to draw a picture of the house or building in which you live.



Turtle Tip: Young children often have trouble using a pen or pencil to draw letters and numbers. If you think about it, there's nothing surprising about this since drawing letters and numbers requires some pretty sophisticated coordination between what the eyes are seeing and what the hand is doing. Logo allows children to practice writing letters and numbers by using the turtle to draw the characters. In this way, they can concentrate on the shape of the characters without having to worry about the manual dexterity needed to draw them with a pen or pencil.

If your child has trouble making the turtle trace the character correctly, suggest "playing turtle." Playing turtle is a standard Logo activity in which the child physically acts out what the commands for the turtle do. At the same time, you can keep a written record of your child's movements and the corresponding Logo commands. These commands can then be entered into the computer to make the turtle draw the desired character on the screen by repeating your child's movements.

For instance, suppose your child is having trouble making the turtle draw the letter T. Place a large piece of paper on the floor with the letter T already drawn on it, and have your child stand at the bottom of the T, facing toward the horizontal line at the top. Have your child imagine that he or she is the turtle and must follow the same path the turtle would take, using the commands for moving the turtle that you've already learned. As your child tries different commands, and physically acts them out, keep track of them by writing them down in order. You can also offer helpful suggestions when your child has trouble. But let your child think out what needs to be done next, and the appropriate command for achieving the result, as much as possible. (With an older child, you can use blank paper and have your child use a pen to draw the figure on the paper while moving around. With a young child of four through six or seven, however, it's better to have the figure already drawn on the paper so your child can concentrate on following the figure rather than having to draw it from memory.)

When the tracing is completed, have your child go back to the starting position at the bottom of the T and follow the commands on the list one at a time. This allows your child (with your help when needed) to check the sequence of commands one final time, either to verify that it gives the desired shape or to identify and correct any problems. The two of you can then go back to the computer and enter this sequence of commands to make the turtle draw the T the same way your child did when playing turtle.

Logo Activity Time

This chapter has introduced you to Logo's turtle and has given you practice in moving it around the screen and drawing pictures with it. As you've done this, you may have noticed that you became better at estimating how much of an angle you needed to turn the turtle in order to face it in a desired direction and how far you needed to move it in order to make it reach a desired point on the screen. These are useful skills for anyone who wants to use Logo's graphics capabilities, and they are especially useful for school-age children who will be learning about the concepts of distance and angles throughout their school years.

The following activity, called MAZE, is designed to help you further sharpen these estimation skills. MAZE is on your Logo Activity Disk. If you haven't created your Activity Disk, you may want to turn to the appendix, where you'll find complete instructions.

You'll have to transfer, or load, MAZE into your computer from the Activity Disk. Place the Activity Disk in the disk drive and close the drive door. Now type:

```
READ "MAZE
```

and press the <RETURN> key. (Notice that you use a beginning quotation mark before MAZE but no end quotation mark after MAZE.)

The red light on the disk drive goes on and the activity contained on the disk under the name MAZE is read into the computer. A series of words is printed on your screen, including MAZE DEFINED, BEGIN DEFINED, and SCREEN DEFINED. MAZE, BEGIN, and SCREEN are the names of procedures that are being read into the computer's memory. (You'll learn what procedures are in Chapter 6.) When the red

light on the disk drive goes off and you see the cursor blinking on your screen again, you'll know the activity is in the computer and is ready for use. To begin the activity, type:

MAZE

and press the <RETURN> key. Your screen should look as shown in Figure 3-11, with a blue background, a yellow maze, and your turtle positioned at the entrance to the maze at the bottom center of the screen, facing upward.

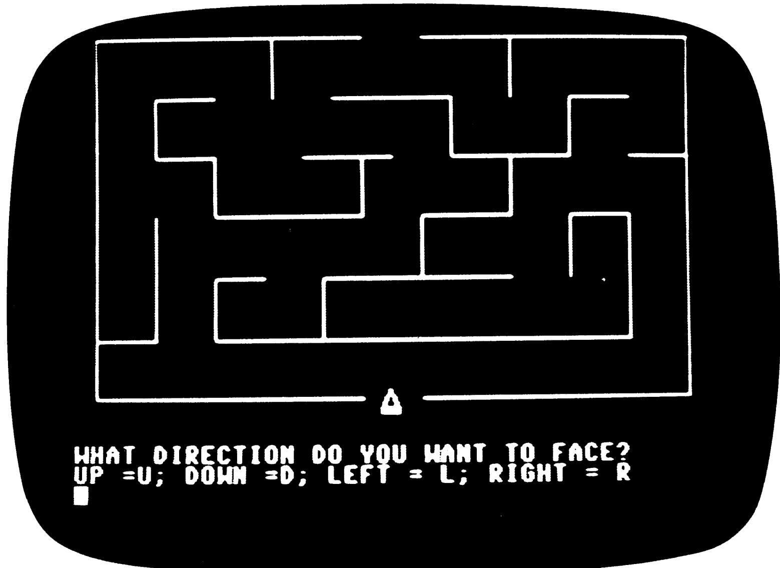


Figure 3-11. The screen at the start of the MAZE activity

In the text area at the bottom of the screen, a message asks which direction you want the turtle to face and instructs you to type U for up, D for down, L for left, and R for right. (You must select one of these four keys even if the turtle is already facing in the correct direction. If it's already facing in the correct direction, use the key that leaves the turtle's direction unchanged.) After you enter one of the four letters and press <RETURN>, the turtle turns to point in the specified direction and you are asked how many steps you

want the turtle to move. After you type a number and press <RETURN>, the turtle moves the specified number of steps and the same two questions are repeated.

The aim of this activity is for you to move the turtle through the maze and out the exit at the top in as few moves—and with as few crashes into the walls—as possible. (Every time the turtle hits a wall, it bounces back ten steps and stops, facing the same direction in which it was moving.) Since the whole idea of this activity is to negotiate the maze in the fewest moves and with the fewest crashes, the ability to accurately estimate the distance in steps to the next turn is important.

For variety, the maze has been designed so that it can be negotiated in more than one way. When your turtle finally exits the maze, the screen is automatically put in the TextScreen format and a message is displayed telling how many moves you took and how many times you hit a wall. You can run the activity again by typing MAZE, or you can continue with any other Logo graphics work by typing DRAW to clear the display screen and put yourself back into the Draw mode.

MAZE remains in the computer's memory, available for you to use, until you turn off the computer. (You'll learn another way to erase the computer's memory of all but the primitive commands in Chapter 4.) Once MAZE is out of the computer's memory, you have to load it back in from the Activity Disk by typing:

```
READ "MAZE
```

before you can use it again.

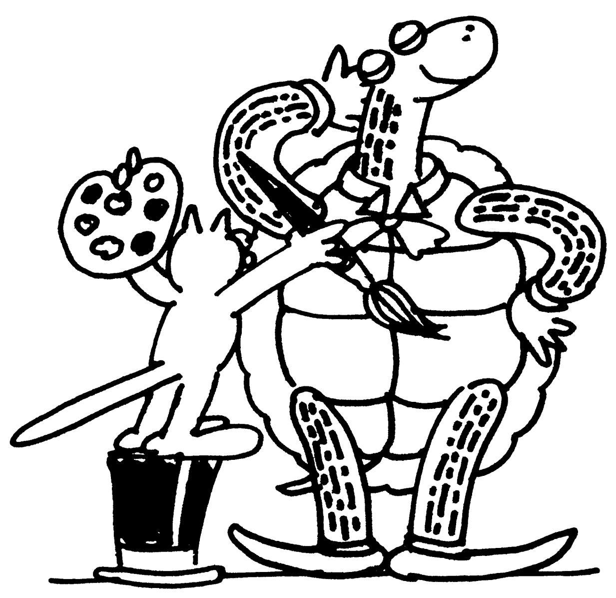
If, for any reason, you want to end the activity before reaching the exit, simply hold down the control key <CTRL> and press the G key at the same time. That ends the activity and you then can type DRAW to clear the screen and put the turtle back at its Home position.

Commands Introduced In This Chapter

As you go along, you'll learn many new commands in addition to the ones you've encountered in this chapter. At the end of each chapter, you'll find a list of the new commands that have been introduced and any abbreviations. Here's the first such list, covering the commands you've learned in this chapter.

Command	*Abbreviation*
DRAW	
NODRAW	ND
FORWARD	FD
BACKWARD	BK
RIGHT	RT
LEFT	LT
CLEARTEXT (<SHIFT> and <CLR>)	
REPEAT	
WRAP	
NOWRAP	
<↑>	
SPLITSCREEN (<f3>)	
FULLSCREEN (<f5>)	
TEXTSCREEN (<f1>)	
PENUP	PU
PENDOWN	PD
CLEARSCREEN	CS

The Colorful Turtle



When you first begin using Commodore 64 Logo, the drawing portion of the screen is gray and the turtle's trail is white. But gray and white aren't the only colors available if you have a color monitor. Commodore 64 Logo allows you to select any one of 16 colors for either the drawing portion of the screen or the turtle's trail.

The 16 colors are represented by the numbers 0 through 15, as given in Figure 4-1. When you want to change colors, you type the number instead of the name of the color.

Figure 4-1. Commodore 64 Logo colors and their numbers

Color	*Number*
Black	0
White	1
Dark Red	2
Cyan (greenish-blue)	3
Purple	4
Dark Green	5
Dark Blue	6
Yellow	7
Orange	8
Brown	9
Light Red	10
Dark Gray	11
Medium Gray	12
Light Green	13
Light Blue	14
Light Gray	15

Whenever you enter the Draw mode, the background color is automatically set for color number 11 (dark gray) and the pen color is automatically set for number 1 (white). These colors are referred to as the default background and pen colors because they are the colors you'll get if you don't choose some other colors.

Once you're in the Draw mode, you can change the background color by typing BACKGROUND (or BG for short), a space, and then the number of the color you want. Similarly, you can change the turtle's pen color by typing PENCOLOR (or PC), a space, and the number of the color you want. For example, one of my favorite color combinations is a yellow

Turtle Trap: The Commodore 64 keyboard has the names of the colors black (BLK), white (WHT), red (RED), cyan (CYN), purple (PUR), green (GRN), blue (BLU), and yellow (YEL) on the fronts of the keys numbered 1 through 8. These color/number combinations are not the same as those shown in Figure 4-1. You use these color/number combinations when you are working in BASIC, not when you are working in Logo. Don't be misled by the names on the keys.

pen (color number 7) on a blue background (color number 6). To change to these colors, type:

```
BACKGROUND 6
PENCOLOR 7
```

or you can type:

```
BG 6
PC 7
```

Now, draw a square or a triangle to see how the color combination looks. As we saw in Chapter 3, the command for drawing a square with sides 50 turtle steps long is:

```
REPEAT 4 [FD 50 RT 90]
```

and the command for drawing a triangle with sides 60 turtle steps long is:

```
REPEAT 3 [FD 60 RT 120]
```

Notice that when you changed the turtle's pen color to yellow, the turtle also became yellow. No matter which color you choose, the turtle is always the same color as the pen.

Without clearing the screen, change the background color to dark red by typing:

```
BACKGROUND 2
```

or by typing:

```
BG 2
```

The yellow figure provides a sharp contrast to the dark red background.

As long as you stay in the Draw mode, your background and pen colors won't change. Even if you clear the screen by typing DRAW or use the <f1>, <f3>, and <f5> keys to switch among TextScreen, SplitScreen, and FullScreen formats, the background and pen colors will stay the colors you pick. But if you switch to the NoDraw mode or if you turn the computer off, when you get back into the Draw mode, the colors revert to their automatically assigned colors, or default values, of dark gray (number 11) for the background and white (number 1) for the pen.

Try changing to each of the background colors to see what they look like. If any of them don't look quite right, you might want to adjust the color controls on your monitor.

You might also want to try various combinations of background and pen colors to see which go together well and which do not. It's helpful to make a list of those combinations you like and may want to use later.

Using an Eraser

In addition to drawing in any one of the 16 pen colors, Logo also allows you to give the pen an eraser. The eraser gives any part of a line that the turtle passes over the same color as the background, making it look as though that part of the line has been erased. You can give the pen an eraser by using one of two commands: PENCOLOR −1 (or PC −1) and PENERASE. For example, suppose you've just used the command:

```
REPEAT 4 [FD 50 RT 90]
```

to draw a square with sides 50 turtle steps long. You can now use the pen's eraser to erase the left side and top of the square. To do this, first type PENCOLOR −1, PC −1, or PENERASE to turn the turtle's pen into an eraser. Then type REPEAT 2 [FD 50 RT 90] to send the turtle back over the left

side and top of the square, erasing the previously drawn lines as it moves over them. Figures 4-2 and 4-3 show the square before and after its left side and top are erased.

Figure 4-2. A square before its left side and top line are erased

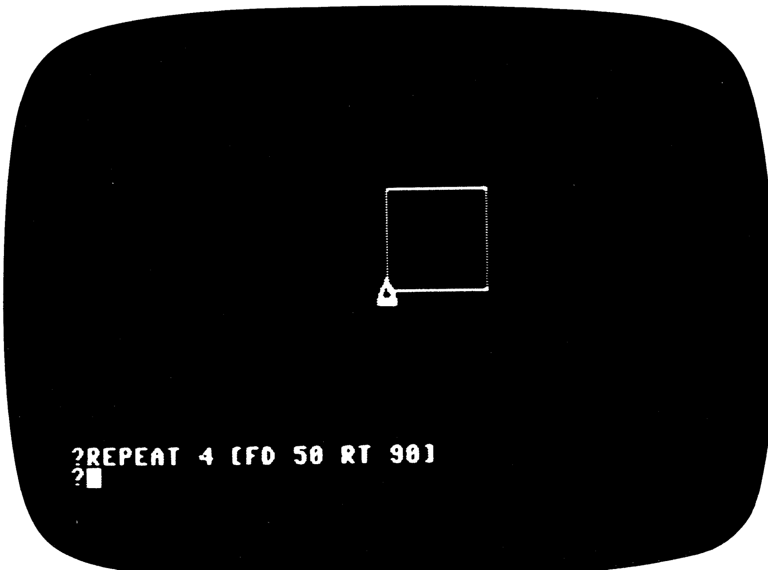
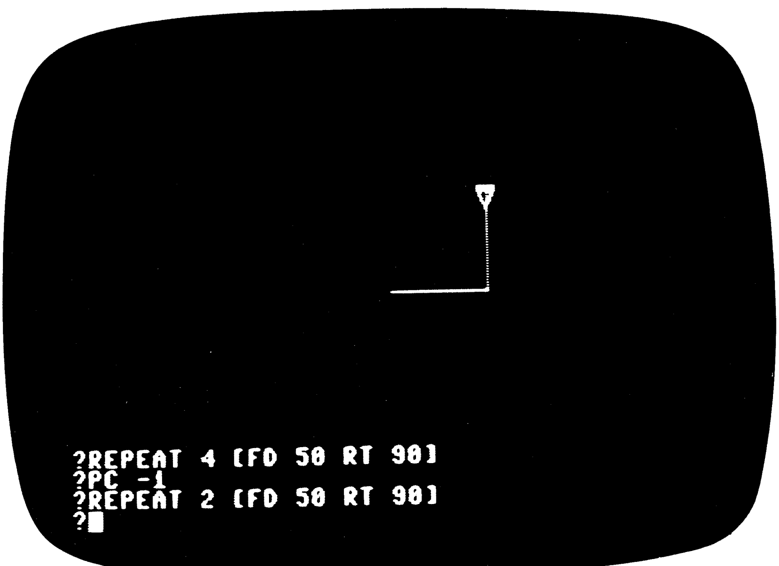


Figure 4-3. A square after its left side and top line are erased



Choosing Colors for The Text Area of Your Screen

The text area of your screen, either in the NoDraw mode or in the SplitScreen and TextScreen formats of the Draw mode, has dark blue (color number 6) letters, numbers, and symbols on a light blue (color number 14) background, unless you change the colors. To change the color of the text, type TEXTCOLOR, a space, and the number of the color you want to use. Likewise, you can change the color of the text background by typing TEXTBG, a space, and the number of the color you want to use. You can select any of the 16 colors. Unlike the background and pen colors, however, the text background and text colors do not revert to their default values when you switch to the NoDraw mode and back to the Draw mode. The only way to set them back to their default values is to select the original colors again using the commands TEXTBG and TEXTCOLOR.

More Things to Do

Try drawing some of the figures you developed in Chapter 3 in new combinations of background and pen colors. You can even switch colors in the middle of a drawing by starting a design in one pen color and then changing to a second pen color. For example, one of your designs consisted of eight squares, with a 45-degree rotation (or turn) between each successive square. To draw the first square and get the 45-degree turn, use the command:

```
REPEAT 4 [FD 50 RT 90] RT 45
```

Repeat the command seven more times with the use of the <↑> key.

After you've drawn the design, use the command PC −1 or PENERASE to give your pen an eraser, and then repeat the design. This results in the design being erased piece by piece in the same order in which it was first drawn.

To obtain squares with alternating colors—for example, red and yellow on a black background—use the following command instead:

```
REPEAT 4 [FD 50 RT 90] RT 90
```

Change the background to black (BG 0) and do four repetitions of this command in dark red (PC 2) to get the first, third, fifth, and seventh squares. Then, change the pen color to yellow (PC 7), rotate the turtle 45 degrees to the right, and repeat this command four more times to get the second, fourth, sixth, and eighth squares. The entire sequence of commands looks as follows:

```
BG 0 PC 2
REPEAT 4 [FD 50 RT 90] RT 90
<↑>
<↑>
<↑>
PC 7 RT 45
REPEAT 4 [FD 50 RT 90] RT 90
<↑>
<↑>
<↑>
```

See if you can draw similar alternating color designs with some of the other patterns you created in Chapter 3.

Another interesting thing to do with Logo's color capability is to draw a shape, then fill it in with a color. For example, suppose you've set your background color to light red (BG 10), your pen color to yellow (PC 7), and you've drawn a square with sides 50 steps long using the command:

```
REPEAT 4 [FD 50 RT 90]
```

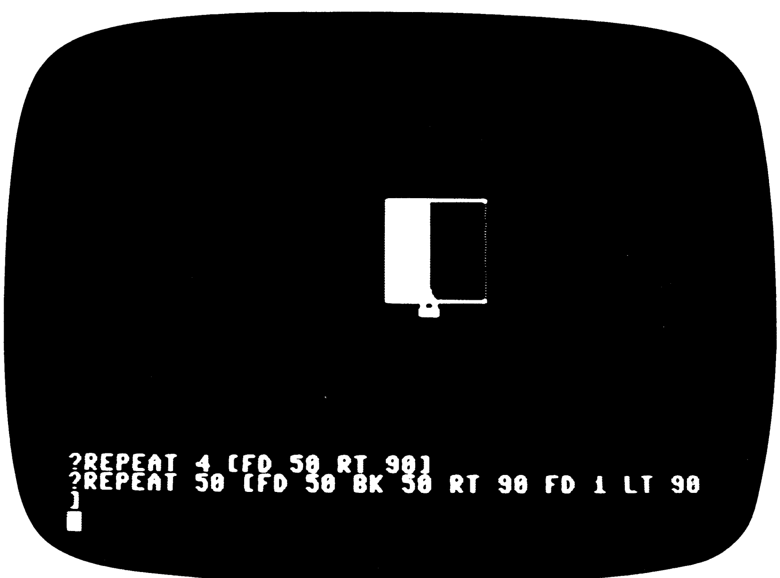
You can fill in the square with yellow by making the turtle move forward 50 steps and backward 50 steps, turn right 90 degrees, move forward one step, turn left 90 degrees,

and repeat these actions 50 times. You fill in the square with color with this command:

```
REPEAT 50 [FD 50 BK 50 RT 90 FD 1 LT 90]
```

Moving the turtle back and forth across the square with a yellow pen fills in the square with yellow, as illustrated in Figures 4-4 and 4-5.

Figure 4-4. Filling in a square



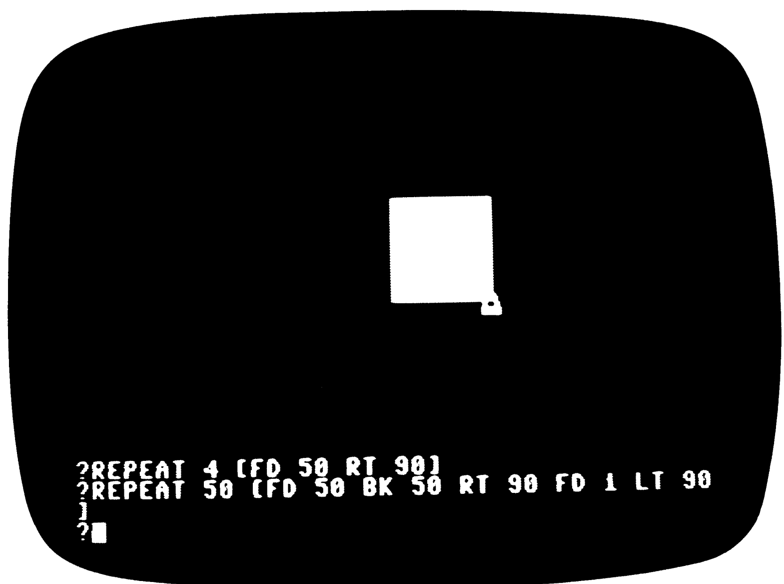


Figure 4-5. The square after it has been filled in

An alternate method of drawing and then filling in the square is to draw successively smaller squares within each other until you end with a square with sides one step long. For example, to color in a square with sides 10 turtle steps long this way, you would use the sequence of commands:

```
REPEAT 4 [FD 10 RT 90]
REPEAT 4 [FD 9 RT 90]
REPEAT 4 [FD 8 RT 90]
REPEAT 4 [FD 7 RT 90]
REPEAT 4 [FD 6 RT 90]
REPEAT 4 [FD 5 RT 90]
REPEAT 4 [FD 4 RT 90]
REPEAT 4 [FD 3 RT 90]
REPEAT 4 [FD 2 RT 90]
REPEAT 4 [FD 1 RT 90]
```

You'll learn a much easier way to write this sequence of commands in Chapter 6, so don't worry about its length now. Just try enough of the commands in this sequence to assure yourself that it actually does color in the square.

Turtle Trap: As you draw shapes in different colors on the screen one after the other, you may notice one line taking on the color of another line that's near or touching it. This is caused by the way in which Commodore 64 colors are generated. It's not a major flaw, but it can be annoying. Just be aware that some of the pictures you'll draw, including some suggested later in this book, may not look exactly as you want them to look (or as described), and that it's due to this phenomenon.

You might want to clear the screen by typing DRAW and then try to draw and color other figures, such as a rectangle and a triangle, the same way you colored in the square. Telling the turtle to move forward and backward across the shape works with the rectangle but you will need to resort to the alternative approach of drawing smaller and smaller figures within each other to color in the triangle. Try it and see what happens.

Changing the Width of The Turtle's Trail

You may have noticed that the lines you've been drawing have been somewhat thin and that when lines of different colors come too close to each other, they sometimes blur and lose their distinct colors. Commodore 64 Logo allows you to draw with thicker lines to produce more distinct figures and less blurring of colors.

When you first enter the Draw mode, Logo is automatically set to draw thin lines, or what is called SingleColor. To draw with thicker lines, type DOUBLECOLOR. To switch back to a thinner line, type SINGLECOLOR. Switching to the NoDraw mode and back to the Draw mode again or switching among different drawing screen formats does not affect your SingleColor or DoubleColor selection.

The one inconvenience of switching from SingleColor to DoubleColor or vice versa is that such a switch automatically

erases any drawing you have on the screen, causes the background and pen colors to revert to their default values, and sets the drawing screen to the SplitScreen format. So, it's important that you decide on SingleColor or DoubleColor before you begin your drawing, not in the middle of your drawing.

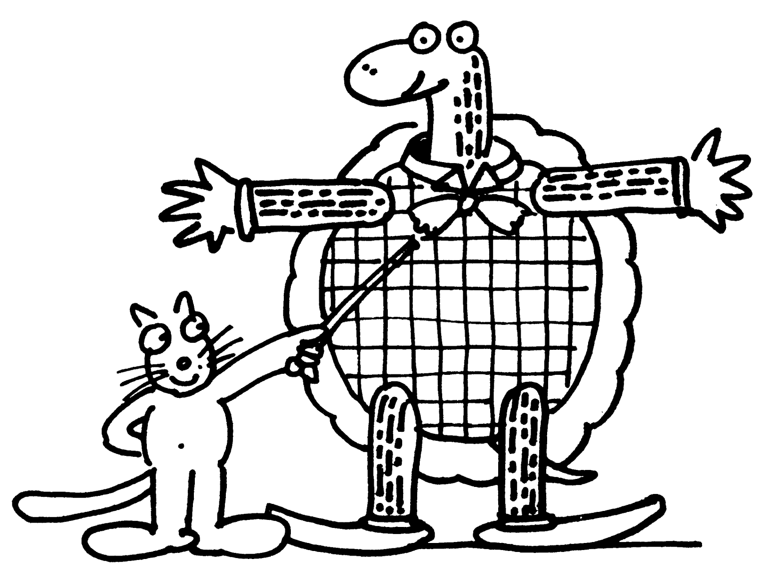
The Invisible Turtle

Logo allows you to hide the turtle from view as it moves about the screen with the command HIDETURTLE (or HT). The turtle continues to obey all your commands, and will leave a trail as it moves if its pen is down, even when it's hidden. You simply won't be able to see the turtle moving. You can make the turtle reappear on the screen by typing the command SHOWTURTLE (or ST). We'll discuss the use of these commands more fully in a later chapter. For now, just try practicing with the HIDETURTLE and SHOWTURTLE commands to see how they work.

New Commands Introduced In This Chapter

Command	*Abbreviation*
BACKGROUND	BG
PENCOLOR	PC
PENERASE	
TEXTBG	
TEXTCOLOR	
SINGLECOLOR	
DOUBLECOLOR	
HIDETURTLE	HT
SHOWTURTLE	ST

5 The Turtle As Grid Hero



Moving the turtle around the screen and drawing shapes with a series of commands, as you did in Chapters 3 and 4, was a laborious process, particularly when you wanted to start a drawing somewhere besides the center of the screen, the turtle's Home position. In this chapter, you'll learn how to move the turtle to any screen location with a single command that uses the screen as a grid, not as a blank area. The mathematical concept behind the

grid, called analytical (or coordinate) geometry, is generally introduced in junior-high or high school and is used throughout the rest of high school, college, and after graduation. So, very young children may not really comprehend the principle involved. But they should be able to follow the instructions.

The command that uses the screen as a grid is SETXY.

Logo's X,Y Coordinate Grid

To understand how SETXY works, think of the screen as a network of horizontal and vertical lines, something like a city street map. The lines are set one turtle step apart, as illustrated in Figure 5-1.

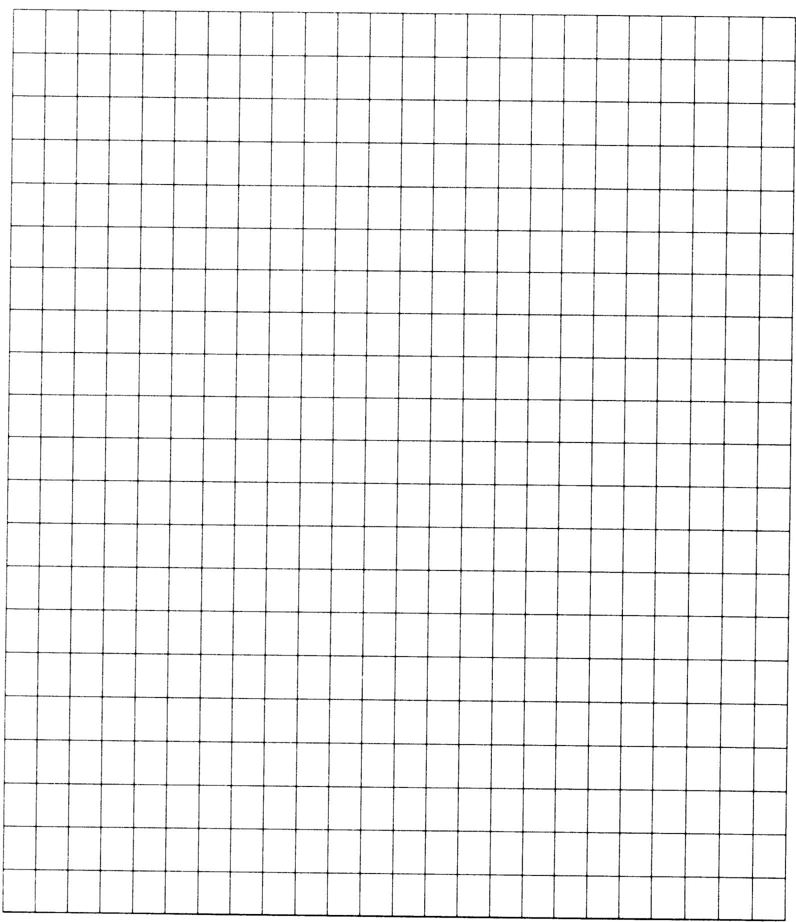
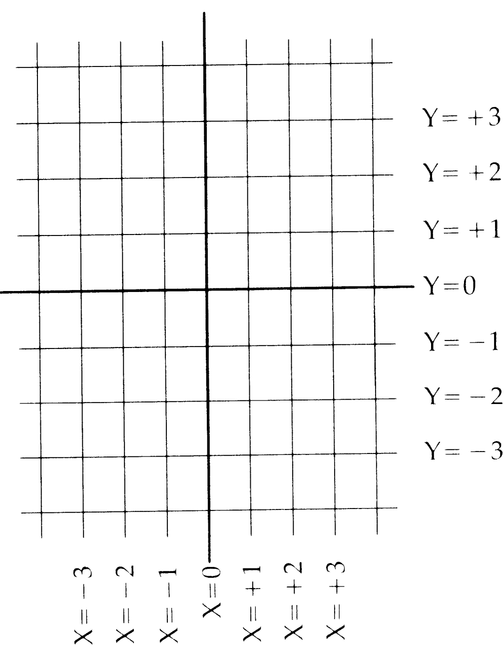


Figure 5-1. A grid similar to the invisible grid on the screen

Each vertical line in this grid is an X coordinate, or position, and each horizontal line is a Y coordinate, or position. The vertical line passing through the center of the screen (the turtle's Home position) is the 0 coordinate for X; the vertical lines to the right of center are positive (1, 2, 3, and so on); and the vertical lines to the left of center are negative (−1, −2, −3, and so on). Similarly, the horizontal line passing through the center of the screen is the 0 coordinate for Y; the horizontal lines above the center are positive (1, 2, 3, and so on) and the horizontal lines below the center are negative (−1, −2, −3, and so on). The X,Y coordinate grid system, as it's called, is illustrated in Figure 5-2.

Figure 5-2. A grid with X and Y coordinates



Using this vertical and horizontal coordinate numbering system, each intersection of a vertical and a horizontal line has an X and a Y number, or value, called its ordered pair of values. The X value in the pair tells you how far to the left or right of center the point is. The Y value in the pair tells you how far below or above center the point is.

When writing about a particular point on a grid, the X and Y coordinate numbers are usually enclosed within parentheses and separated by a comma. For example, if you start at the center of the screen, the X,Y coordinates for the point five steps to the right and five steps up are (5,5); the X,Y coordinates for the point five steps to the left and five steps up are (−5,5); the X,Y coordinates for the point five steps to the left and five steps down are (−5, −5); and the X,Y coordinates for the point five steps to the right and five steps down are (5, −5), as you can see in Figure 5-3.

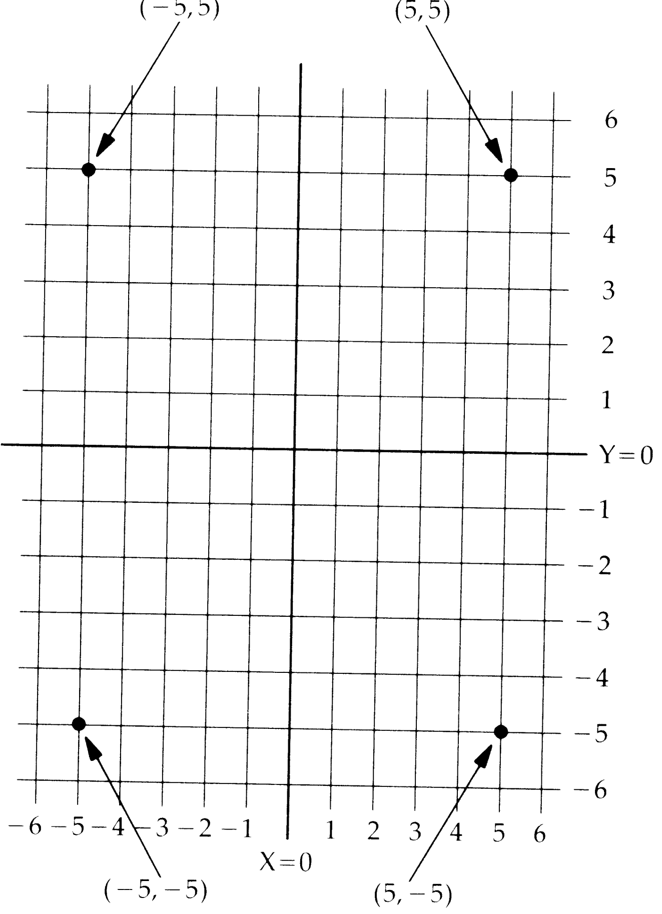


Figure 5-3. Four points on an X, Y coordinate grid

Positioning the Turtle on the X,Y Grid

Suppose that you want to move the turtle to the screen position with an X coordinate of 100 and a Y coordinate of 100. All you have to do is type SETXY, a space, 100 (the X coordinate of the point), another space, and 100 (the Y coordinate of the point). The turtle immediately moves in a straight line from wherever it is on the screen to the (100,100) location. If its pen is down when you give the command, it leaves a trail as it moves.

For practice, clear the screen with DRAW and then move the turtle to the position where both the X and Y coordinates are 100, by typing:

```
SETXY 100 100
```

Now, move the turtle to the position where the X coordinate is −100 and the Y coordinate is 50, by typing:

```
SETXY −100 50
```

Your screen now looks as shown in Figure 5-4.

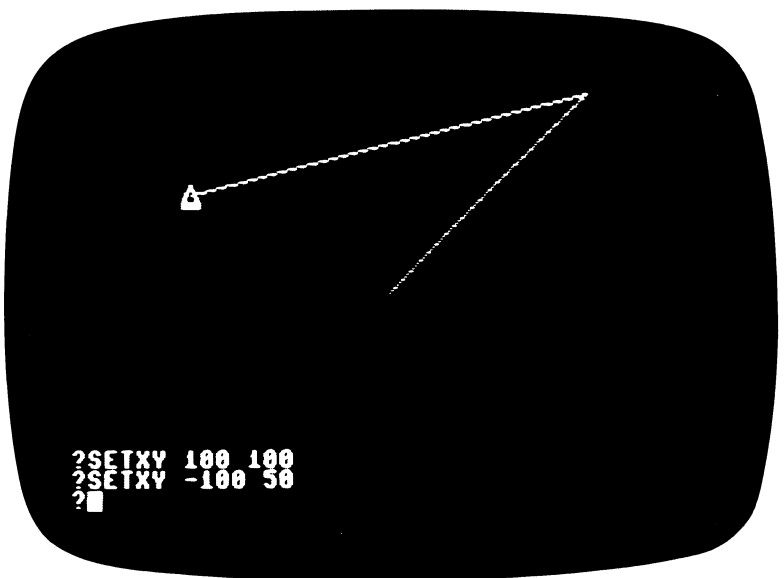


Figure 5-4. The turtle moved with the SETXY command

As I just mentioned, the turtle leaves a trail, just as it would if you moved it with FORWARD and BACKWARD. If you want to move the turtle from one spot to another using SETXY without leaving a trail, you must first type PENUP (or PU). Once you've reached your destination, you can then lower the pen, using PENDOWN (or PD).

Notice that the SETXY command changes the turtle's position but not the direction in which it is pointing. In other words, if it is pointing 45 degrees to the right when you give the SETXY command, it will still be pointing 45 degrees to the right at the new location.

When the Y coordinate in the SETXY command is negative, you need to type an open parenthesis before and a close parenthesis after the negative number. To understand why this is necessary, consider the command SETXY 100−50.

You might think that this command is telling the computer to send the turtle to the location where the X coordinate is 100 and the Y coordinate is −50 (that is, to a position that is 100 turtle steps to the right of the screen's center and 50 turtle steps below the screen's center.) The computer, however, sees 100−50 as a subtraction problem rather than as two separate numbers. So, it takes the result of this subtraction problem, 100−50=50, as the first number following SETXY (the X coordinate) and looks for the second number (the Y coordinate). It doesn't see a second number, so it displays the error message SETXY NEEDS MORE INPUTS in the text portion of the screen.

Enclosing the negative Y coordinate value within parentheses as follows tells the computer that 100 and −50 are two separate coordinates and not a subtraction problem:

```
SETXY 100 (−50)
```

For simplicity, from now on enclose all negative coordinates that occur in the SETXY command within parentheses whether they refer to the X coordinate or the Y coordinate.

Turtle Tip: Grid systems, such as the one used by the SETXY command, represent an important development in the history of mathematics. Use of horizontal and vertical coordinate lines to represent geometric points with ordered pairs of numbers is the concept underlying analytic, or coordinate, geometry and is the creation of the French mathematician René Descartes, who lived from 1596 to 1650.

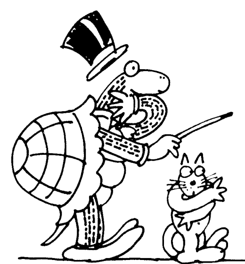
Analytic geometry is important in mathematics because it sets up a simple relationship between points (geometric objects) and ordered pairs of numbers (algebraic objects), allowing problems and questions in either geometry or algebra to be studied and solved using the tools and techniques of both subject areas.

The X,Y coordinate grid, and its use in the development of analytic geometry, is usually first introduced at the junior-high or high-school level. Once it's been introduced, however, it keeps reappearing in mathematics, physics, and engineering classes in high school and college.

Because of the importance of coordinate systems in school mathematics, it's important to expose school-age children to coordinate systems. The SETXY command in Logo provides this exposure. For this reason, you should encourage your children to use the SETXY command to move the turtle around the screen whenever it seems appropriate to do so.

While parentheses are really only necessary for negative Y coordinates, it doesn't hurt for you to put them around negative X coordinates as well. Putting parentheses around all negative numbers assures that it's done in those instances when it's necessary.

Logo allows you to move the turtle either to a new X coordinate without changing the Y coordinate—that is, move it horizontally—or to a new Y coordinate without changing the X coordinate—that is, move it vertically. To move the turtle horizontally, type SETX, a space, and the new X coordinate; to move it vertically, type SETY, a space, and the new Y coordinate. For example, SETX 75 moves the turtle horizontally to the screen position with its current Y coordinate but with the X coordinate 75. SETY 60 moves the turtle vertically to the screen position with its current X coordinate but with the Y coordinate 60.



Turtle Trap: The Commodore 64 computer has a key labeled <HOME> at the upper right of the keyboard next to <DEL>. However, this key does not make the turtle move to its Home position at the center of the screen. In order to move the turtle to its Home position on the screen without erasing any screen pictures, you need to type the word HOME one letter at a time; you cannot simply press the <HOME> key. Use of the <HOME> key will be discussed in Chapter 7.

Going Home

Another command that moves the turtle to a particular spot on the screen from anywhere else is the command HOME. Typing HOME is similar to typing SETXY 0 0 in that it sends the turtle straight to its Home position at the center of the screen. The difference is that, while SETXY 0 0 does not change the direction in which the turtle is pointing, HOME does. HOME sends the turtle in a straight line to position (0,0)—leaving a trail as it goes if its pen is down when the command is given—and leaves it pointing straight up, whether or not it was pointing straight up before it moved.

Rotating the Turtle

Two commands—SETHEADING and SETHEADING TOWARDS—leave the turtle in the same location but change the direction in which it is pointing.

To rotate the turtle a specified number of degrees from straight up, simply type SETHEADING (or SETH), a space, and the number of degrees to the right or left you want the turtle to point. The turtle's Home direction, pointing straight up, is 0. A positive number turns the turtle that number of degrees to the right of straight up. A negative number turns

the turtle that number of degrees to the left of straight up. For example, the command:

```
SETHEADING 135
```

which can also be typed:

```
SETH 135
```

has the same effect as making the turtle point straight up (if it wasn't already) and then turning it 135 degrees to the right. The turtle actually turns immediately to a direction that is 135 degrees to the right of straight up, but the result is the same as if it were pointed straight up first and then rotated 135 degrees to the right.

Using the commands SETXY and SETHEADING together immediately moves the turtle to any position on the screen, pointing in any direction.

To rotate the turtle so that it points toward a particular spot on the screen, type SETHEADING TOWARDS (or SETH TOWARDS), a space, and a pair of numbers representing the X,Y coordinates of the spot you want the turtle pointing toward. For example, the command:

```
SETHEADING TOWARDS 100 50
```

which can also be typed:

```
SETH TOWARDS 100 50
```

leaves the turtle at its present position, but turns it so that it is pointing toward the screen position (100,50) (unless, of course, it was already pointing in that direction).

More Things to Do

For the following activities, you may want to use the <f5> key for the FullScreen format, since you'll be sending the turtle all over the screen and you don't want to lose sight of it, or the bottom parts of the figures it's drawing, when it's near the bottom of the screen.

Use the SETXY command with Y as 0, but larger and larger values for X to discover the largest X coordinate on the right side of the screen. Then, do the same thing with Y as 0 and negative values of X to discover the smallest X coordinate on the left side of the screen; with X as 0 and positive values of Y to discover the largest Y coordinate at the top of the screen; and with X as 0 and negative values of Y to discover the smallest Y coordinate at the bottom of the screen. Make a chart showing the grid, with the coordinates of these outermost points for later use in positioning the turtle at desired locations.

Draw a square with sides 100 steps long, centered at screen location (0,0) without using the commands FORWARD, BACKWARD, RIGHT, or LEFT. As shown in Figure 5-5, this square has its corners at the points with coordinates (−50,50), (50,50), (50, −50), and (−50, −50). Use SETXY with PENUP to position the turtle at any one of these corners. Then, after using PENDOWN, use SETXY (or SETX or SETY) four more times to move the turtle to each corner in turn. As it moves, the turtle leaves a trail connecting the corners and this trail forms the square shape.

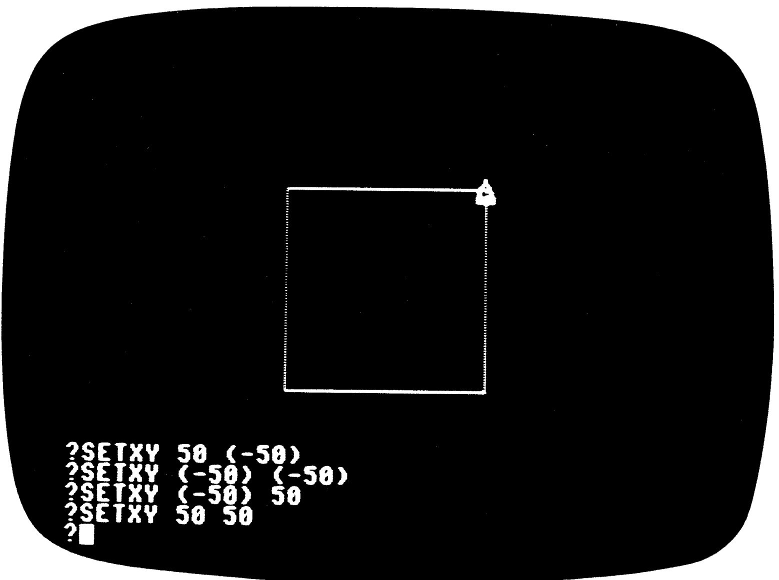


Figure 5-5. A square drawn using the SETXY command

Once you've successfully drawn the square with sides 100 steps long using only the SETXY, SETX, and SETY commands, use the same procedure to draw squares with sides 75, 50, and 25 steps long within it. Your final picture of four squares, one inside the other and all with the Home position as their center, should look as pictured in Figure 5-6. If you want, you can draw each square in a different color.

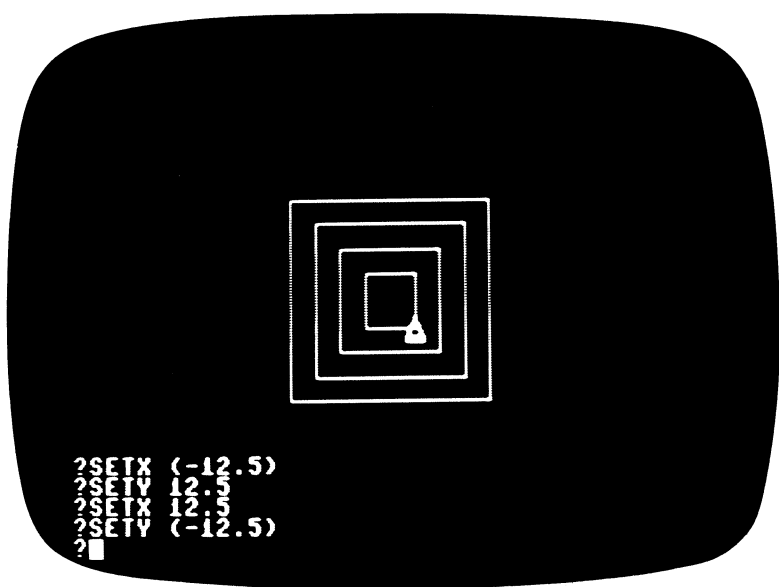


Figure 5-6. Four squares drawn one inside another using SETXY, SETX, SETY

Finally, clear the screen using DRAW and use SETXY, SETX, and SETY to draw four rectangles, one inside the other, centered at the Home position. Make each rectangle twice as long as it is high, with the largest rectangle being 50 steps high and 100 steps long.

Logo Activity Time

In order to move the turtle directly to a desired location on the screen using the SETXY command, it's necessary to be able to estimate the X and Y coordinates of that screen location. To practice these estimation skills, use the activity called FIND.ME on your Logo Activity Disk.

To use FIND.ME, place your Activity Disk in the disk drive and close the drive door. Now type:

```
READ "FIND.ME
```

The red light of the drive goes on and you hear a low whirring noise to indicate that the drive is working. A series of words are printed on your screen:

```
FIND.ME DEFINED
SETSCREEN DEFINED
HIDE DEFINED
DASH.AROUND DEFINED
```

FIND.ME, SETSCREEN, HIDE, and DASH.AROUND are the names of procedures that are being read into the computer's memory. (You'll learn what procedures are in Chapter 6.) When the drive's red light goes off and the blinking cursor square reappears on the screen, the FIND.ME activity is in the computer and is ready for you to use. Now, take your Activity Disk out of the disk drive and put it away in its protective envelope for safekeeping.

Once the activity is in the computer, you can play it by typing:

```
FIND.ME
```

Your screen switches to the Draw mode in the SplitScreen format with a blue drawing area. The turtle quickly draws a horizontal and a vertical line through the center of the screen in yellow. Then, the turtle dashes around the screen, finally coming to rest at a randomly selected location. Your display screen now looks something like the illustration in Figure 5-7.

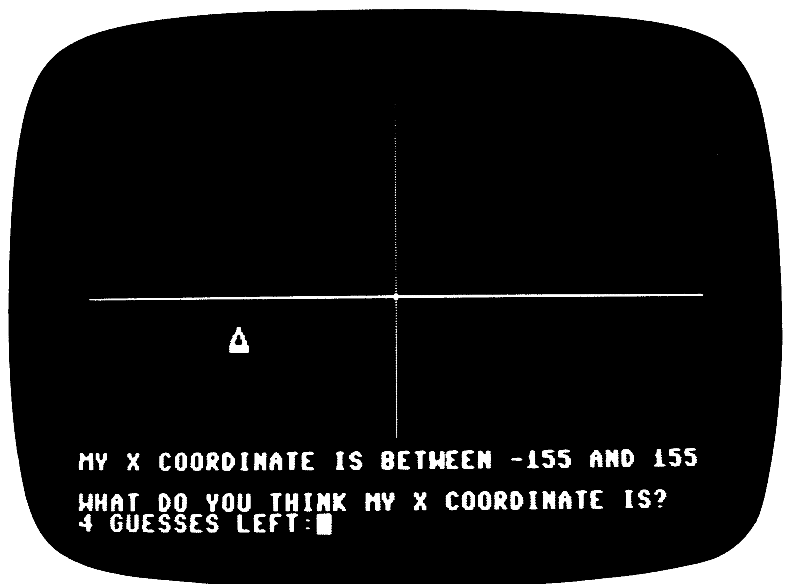


Figure 5-7. A screen at the start of the FIND.ME activity

The X coordinates stretch from −155 on the left to 155 on the right and the Y coordinates reach from −70 on the bottom to 130 on the top. In the text area at the bottom of the screen is a message, reading:

MY X COORDINATE IS BETWEEN −155 AND 155
WHAT DO YOU THINK MY X COORDINATE IS?
4 GUESSES LEFT:

Based on where the turtle is located within the rectangle, enter your guess for the X coordinate and press <RETURN>. A similar message then appears asking you to guess the Y coordinate. Enter a guess for Y and again press <RETURN>.

If you don't guess correctly the first time, a yellow rectangle is drawn with your guess as one of its corners. Another message asks for a new X guess. But this time, one of the numbers is your first guess. After you make your second X guess, a message asks for a new Y guess, again with one of the numbers from your first guess.

For instance, suppose the turtle is located at a screen position having an X coordinate of 90 and a Y coordinate of 80. If your first guess is 50 for X and 75 for Y, the second pair of messages will ask for an X coordinate between 50 and 155 and a Y coordinate between 75 and 130. If your second guess is 100 for X and 90 for Y, the next two messages will ask for an X coordinate between 50 and 100 and a Y coordinate between 75 and 90.

If you guess either the X coordinate or the Y coordinate correctly, you're told that guess was correct and the rest of the questions pertain only to the remaining coordinate. If you guess both coordinates, the turtle draws a star at that location, and the screen switches to the TextScreen format, with a summary of the results displayed. If you don't guess both coordinates in four attempts, the activity is over. The screen switches to the TextScreen format. You're told the turtle's exact X and Y coordinates and how far off your closest guess for each coordinate was. Your total error score is the sum of your X error and your Y error. The object of the game is to get as small an error score as possible, with a zero error score representing correct guesses for both X and Y.

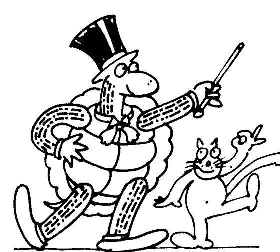
You can continue to other Logo work by typing NODRAW or DRAW to clear the screen. Or, you can play the activity again by typing:

```
FIND.ME
```

You can stop the activity at any point while you're playing it by holding down the control key <CTRL> and pressing the G key at the same time.

The FIND.ME activity remains in your computer until you either turn off the computer or explicitly clear the computer's memory of all but its primitive Logo commands. (You'll learn how to do that in the next chapter.) If you turn off the computer or clear its memory, you'll have to once again load the activity into the computer from the Activity Disk by typing:

```
READ "FIND.ME
```



Turtle Tip: The More Things to Do section earlier in this chapter is a first step in acquainting your child with coordinate grid systems. Another step is provided by FIND.ME. FIND.ME serves three distinct purposes.

First, and most obvious, it gives children practice in estimating the X and Y coordinates of various locations on the screen so that they will be better able to use SETXY to position the turtle at a desired location when they want to draw pictures or do other Logo activities introduced later in this book.

Second, children will become familiar with X and Y coordinate systems, and will be comfortable with them when they are introduced in school.

Third, it exposes your children to signed numbers—that is, negative, positive, and zero. It also introduces them to the important mathematical topic of the ordering of signed numbers—that is, determining when one signed number is smaller or larger than another. For example, if your child is told that the X coordinate of the turtle is between −50 and 35 and makes a next guess of −75, a message appears on the screen saying the last guess was too small and asking for another try. (A guess that's outside the specified range is not counted as one of the child's four guesses.) Your child soon realizes that for a guess to be acceptable to the computer, it must be between the two given values, and learns through trial and error what it means for one signed number to be smaller or larger than another.

Keep in mind that, while the ostensible purpose of the activity is to "win" by getting as low an error score as possible, getting a perfect score of zero is really as much a matter of good guessing as anything else. It's impossible to tell exactly where the turtle is by sight, since one step is so small compared to the entire screen. The real object is just to become familiar with the way the coordinate system works and to use this understanding to improve from one guess to the next, getting close to the exact position.

Finding Out Where the Turtle Is and What It's Doing

Sometimes, you'd like to know certain things about the turtle's status, such as its X and Y coordinates, its heading (or direction), and whether its pen is up or down. You can obtain this and other information using the command DRAWSTATE.

When you type DRAWSTATE, the computer displays the message RESULT: followed by brackets containing nine items

of information about the current drawing status. For example, typing DRAWSTATE might produce a display message something like this:

```
RESULT: [TRUE TRUE 11 1 DRAW SINGLECOLOR
 SPLITSCREEN 14 6]
```

Any time you type DRAWSTATE, the nine items of information, in order of display, are:

1. The word TRUE if the turtle's pen is down, or FALSE if the pen is up.
2. The word TRUE if the turtle is showing, or FALSE if the turtle is hidden.
3. The background color of the drawing screen.
4. The turtle's pen color.
5. Whether you are in the Draw or NoDraw mode.
6. Whether the turtle's trail is SingleColor or DoubleColor.
7. Whether you are in FullScreen, SplitScreen, or TextScreen format.
8. The text background color for the text area of your screen.
9. The color of the text printed in the text area of your screen.

So, in the example just given, the turtle's pen is down, it's showing, the background color is dark gray, the pen color is white, the screen is in the Draw mode, the trail width is SingleColor, the screen is in the SplitScreen format, the text background is light blue, and the text is dark blue. You can also request the current status of any particular one of these

nine items by typing the command ITEM, a space, the particular item number you want, a space, and the command DRAWSTATE. For example, the command:

```
ITEM 3 DRAWSTATE
```

tells the computer to give you the current background color number, since item 3 on the list is background color. The command:

```
ITEM 4 DRAWSTATE
```

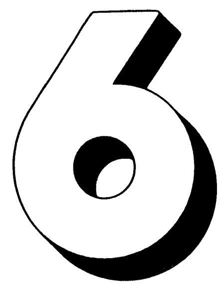
tells the computer to give you the current pen color number, since item 4 on the list is pen color.

You can find out the turtle's X coordinate by typing XCOR, the turtle's Y coordinate by typing YCOR, and the direction the turtle is heading by typing HEADING.

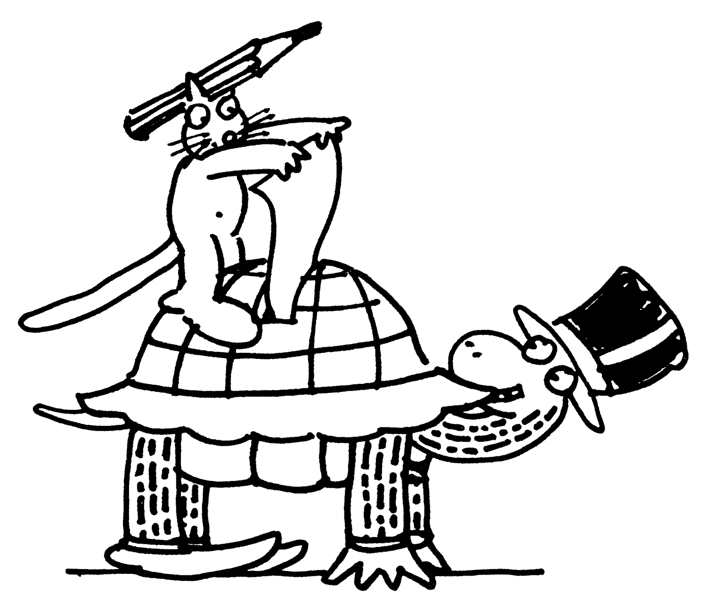
Now that you've had some practice moving the turtle around the screen and changing the background and pen colors, let's go on to the next chapter and the topic of extending our Logo vocabulary of commands using procedures.

New Commands Introduced In This Chapter

Command	*Abbreviation*
SETXY	
SETX	
SETY	
SETHEADING	SETH
SETHEADING TOWARDS	SETH TOWARDS
HOME	
DRAWSTATE	
ITEM DRAWSTATE	
XCOR	
YCOR	
HEADING	



Writing Your Own Commands



Starting with a simple vocabulary of built-in, primitive commands, Logo allows you to put together new commands in terms the computer already understands. Each of these new commands is called a procedure and is a set of existing instructions that you group together under a unique name to have the computer do something new. A procedure name can be as short as one letter or as long as you need, and once you define and name a procedure,

the name alone tells the computer to carry out every instruction defined by the procedure.

Building a procedure is important because it parallels the way you learn—moving from the simple to the complex. And the concept of a procedure is the key to Logo itself, because built-in commands allow you to use the language, while procedures help you master it.

When you turn your computer off, both the built-in, primitive commands and the procedures that you've written are forgotten by the computer. The difference between them, however, is that the primitive commands are still contained on the Logo language disk and can be easily read into the computer any time you want to use them, while the procedures you've written are not contained on the Logo language disk and have to be typed in again. You can avoid losing the procedures you've written and having to retype them by saving them on a separate disk that has been initialized, or prepared, to receive information from a Commodore 64 system. These procedures can then be read, or transferred, from the disk back into the computer whenever you want to use them, the same way you read the primitive commands back into the computer from the language disk.

Writing a Procedure

You're now ready to write a procedure. Let's start with a simple one. Let's write a procedure called SQUARE that tells the turtle to draw a square with sides 50 steps long.

Entering the Edit Mode

First, type the words TO SQUARE after the question mark in the text area of your screen.

The word TO tells the computer you're going to tell it how to do something, as if you were about to say, "To drive a car, you. . . ." But, instead of driving a car, you're going to tell it how to SQUARE. Now, press <RETURN>.

Your screen switches to what's called the Edit mode, a gray background with white letters, as you can see in Figure 6-1. The words TO SQUARE appear at the top of the screen. That's the cursor blinking on and off below the T of TO, waiting for you to continue typing on the next line. (Don't worry about the words at the bottom of the screen. We'll get to that in a minute.)

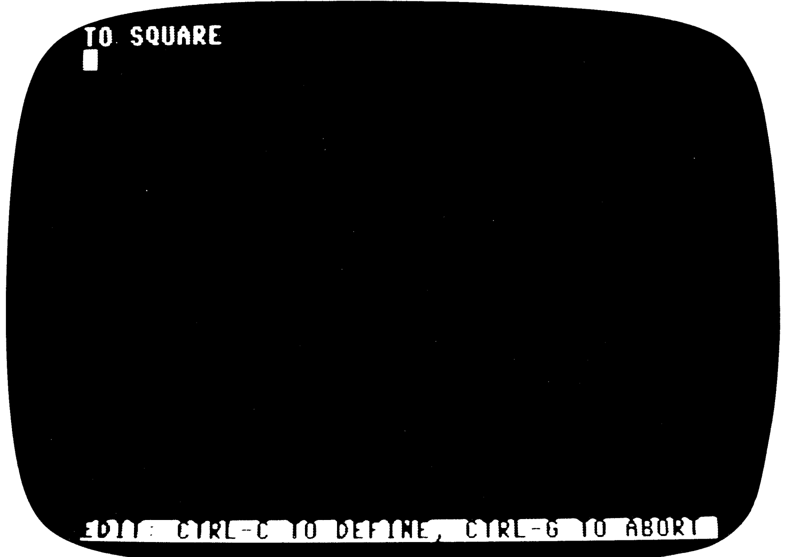


Figure 6-1. The screen in the Edit mode, as you start to type the SQUARE procedure

The Edit mode is used to define new procedures or to edit, or modify, previously defined procedures. Logo automatically switches to the Edit mode whenever you indicate you want to define a new procedure by typing the word TO followed by a space and a name, or whenever you indicate you want to edit a procedure by typing the word EDIT (or ED for short) followed by a space and the procedure's name.

Defining a Procedure

Now, you're ready to type in what you want the computer to do when you use the SQUARE procedure. You can put all of the commands on one line, separated by spaces, or you can put commands on different lines by pressing <RETURN> when you want to start a new line. It's a good idea not to put too many separate commands on the same line. If you write each command on a separate line, it's easier to keep track of the procedure and change parts of it later.

If you come to the right edge of the screen while you're typing, just keep typing. The computer automatically inserts an exclamation mark at the edge of the screen to remind you the command is taking more than one display line, and automatically continues the command on the following line.

When you've finished writing the procedure, type the word END on the last line. If you don't, the computer does it for you automatically. However, it's a good habit to get into. After you've gained some experience with writing procedures, there may be times when you might want to define more than one new procedure at a time while you're in the Edit mode. In such a case, the END statement tells the computer where one procedure ends and the next one begins.

Following are two different ways of writing the procedure for drawing a square. Both methods are correct, although the second procedure is somewhat shorter and more efficient than the first.

```
TO SQUARE
 FD 50 RT 90
 FD 50 RT 90
 FD 50 RT 90
 FD 50 RT 90
END
```

```
TO SQUARE
 REPEAT 4 [FD 50 RT 90]
END
```

When you type lines of commands, they all begin at the far left of the screen. However, every time you come back to look at them or change them, they appear as you see here, with all of the lines except the procedure name and the END statement indented one space. Logo indents automatically to help you distinguish the commands from the TO and END statements.

The SQUARE procedure is relatively short and can be displayed in its entirety on the Edit mode display screen. If you define a much longer procedure, however, you'll notice that when the Edit mode display screen is completely filled, the lines of text at the top of the screen seem to disappear as new lines of text are typed in at the bottom. But don't worry. Even though you can't see the lines that have disappeared, the computer still remembers them and treats them as part of the procedure. In fact, you can even use the up-and-down <CRSR> key to bring lines of text that have disappeared off the top of the screen back into view so that you can read them over and, if you want, make corrections or modifications. We'll see how to do this in the next section.

Correcting Mistakes in the Edit Mode

To correct mistakes while typing in the Edit mode, you can use any of several keys. As you may recall, the <DEL> key moves the cursor one position to the left and erases the character in that position. If you hold the <DEL> key down, the cursor continues to move left, erasing as it goes, until you release it.

Two additional helpful keys are the keys at the bottom right of the keyboard marked <CRSR> with arrows on them. These <CRSR> keys let you move the cursor left or right, up or down. Unlike the <DEL> key, the <CRSR> keys move the cursor to where you want to make a change without altering the text the cursor moves across. In Logo, the

left-and-right <CRSR> key can also be used when you are typing in commands outside of the Edit mode, but the up-and-down <CRSR> key cannot. This is because you type only one line of commands at a time, and as soon as you finish typing the line and press <RETURN>, those commands are carried out and forgotten. Since the up-and-down <CRSR> key is used only to move the cursor from its current line to a preceding or following line, it serves no purpose when typing in commands outside of the Edit mode.

Pressing the <CRSR> keys moves the cursor in the directions indicated by the arrows on the bottoms of the keys. If you want to move the cursor in the opposite directions, as indicated by the arrows on the tops of the keys, hold down the <SHIFT> key at the same time as you press one of the <CRSR> keys. If the cursor is at the beginning of a line and you try to move it farther to the left with the left <CRSR> key, it jumps to the end of the previous line, as if the two lines were connected. Similarly, if the cursor is at the end of a line and you try to move it to the right, it jumps to the beginning of the next line.

If the procedure you want to edit is so long that it can't all be displayed on the screen at one time, you can use the up-and-down <CRSR> key to bring the part you want to correct or modify into view. If you want to bring back into view an early part of the procedure that has disappeared off the top of the screen, simply use the up <CRSR> key to move the cursor to the top line of the screen, and then press the up <CRSR> key one additional time. This causes the text that's showing to move down half a screenful, and brings back into view the last several lines that disappeared off the top of the screen. If you want to bring additional lines back onto the screen, just repeat this process. To move the text up half a screenful so that the more recently typed text is brought back from the bottom of the screen, simply use the down <CRSR> key to move the cursor to the bottom line of text and then press the down <CRSR> key one additional time.

To see how these keys are used, suppose you accidentally put a 5 instead of a 4 in the REPEAT command in the second version of SQUARE:

```
TO SQUARE
 REPEAT 5 [FD 50 RT 90]
END
```

To correct this mistake, use the <CRSR> keys to position the cursor just to the right of the incorrect character 5, as shown in Figure 6-2. Then, press the <DEL> key to remove

Figure 6-2. Positioning the cursor to the right of the incorrect character

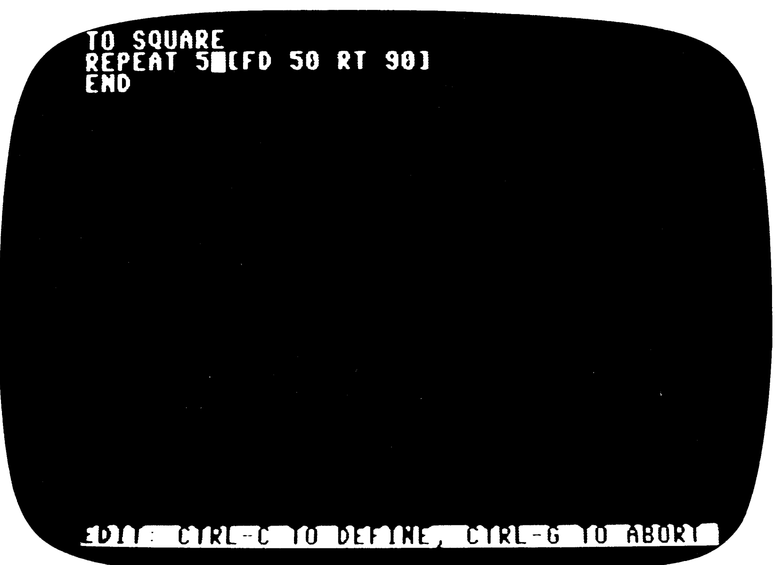
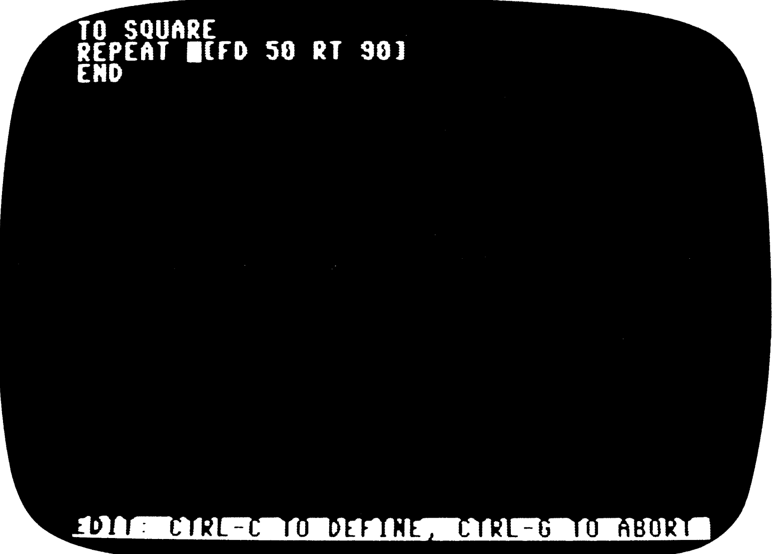


Figure 6-3. Deleting the incorrect character



the incorrect character 5, as shown in Figure 6-3. Finally, press the key marked 4 to insert the correct character where the incorrect one had been, as shown in Figure 6-4. Now, use the <CRSR> keys to move the cursor back to where you were working when you discovered the error.

Figure 6-4. Inserting the correct character

Once the procedure is entered (and, if necessary, corrected), you're ready to tell the computer to add this new command and its definition to Logo's vocabulary. Do this by holding down the <CTRL> key and pressing C at the same time. This is known as defining a procedure. Just to remind you that you need to press <CTRL> C or <CTRL> G, the bottom of the Edit mode screen always displays the message:

EDIT: CTRL-C TO DEFINE, CTRL-G TO ABORT

(You'll learn what the second part of this reminder, CTRL-G TO ABORT, means in Chapter 7.)

When you press <CTRL> C your screen automatically switches to the NoDraw mode and displays the message:

PLEASE WAIT...

After a few seconds (the longer the procedure, the longer the wait), the statement SQUARE DEFINED appears on the screen, with a question mark and the cursor on the line below, telling you that SQUARE has been defined and added to the Logo vocabulary, as shown in Figure 6-5.

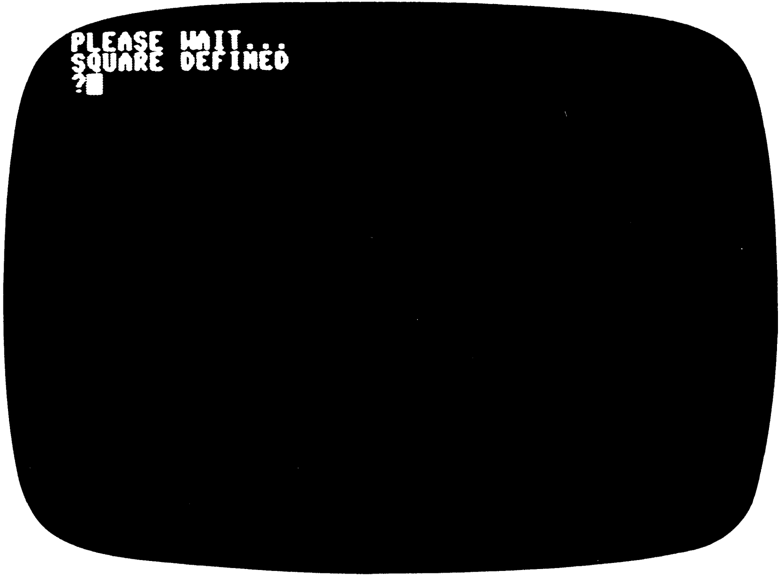


Figure 6-5. The screen after SQUARE is defined

Using a Procedure

Once you have defined a procedure as you have just done with SQUARE, you can use it just as you would use any Logo built-in, or primitive, command. To test the command, type DRAW to enter the Draw mode and then type SQUARE. The turtle draws the square just as you instructed it to do in the procedure. In fact, the turtle draws the square wherever it happens to be when you give the SQUARE command.

If the procedure doesn't work or doesn't do exactly what you wanted it to do, you may need to go back into the Edit mode with the SQUARE procedure displayed and make some corrections. The process of editing a previously defined procedure will be described and illustrated in detail in the next chapter. But, for the time being, you can make minor changes and corrections fairly easily by typing:

```
EDIT SQUARE
```

or by typing:

```
TO SQUARE
```

Typing either one of these commands puts you back into the Edit mode with the SQUARE procedure displayed on your screen. Read through the commands on your screen one at a time and compare them with the procedure listings for SQUARE given earlier. Try to determine why the procedure didn't do what you wanted. Once you've located the problem, use the <CRSR> keys and then the <DEL> key to remove any incorrect characters. Insert any missing characters just as you would if you were defining SQUARE. When you've made all of your corrections, press <CTRL> C to exit the Edit mode, with the changes you've made in the SQUARE procedure implemented. Now try typing SQUARE again to see if your editing has corrected the problem. If the procedure still doesn't work, go back into the Edit mode and try again.

Turtle Tip: Using procedures to draw pictures and create colorful designs with Logo's turtle graphics is enjoyable, but it also serves an important educational purpose. Logo was designed so that children could use its built-in, primitive commands as building blocks to define new commands, just as they progress from the simple to the more complex in the learning process. As children learn to use procedures, they use the mental process of combining simple ideas and concepts to form more complex ones.

In the chapters that follow, you'll learn many new commands that you can incorporate into procedures in order to make the results more efficient, pleasing, or useful. Each time a new command is introduced, encourage your children to experiment with it by using it in procedures. The benefits—intellectual growth and awareness—are well worth the time and effort expended. And the satisfaction your children will feel when a procedure actually works will increase their sense of self-worth and encourage them to try again the next time a challenging problem arises.

Important Points About Procedures

Before going further, a few points need to be made concerning the definition and use of procedures.

First, when you defined the SQUARE procedure, recall that you typed the words TO SQUARE and that these same two words appeared as the first line of the procedure. However, when you used the procedure to draw a square, you left out the word TO and simply typed SQUARE. The word TO is usually used when you first define the procedure. It tells the computer you want to define a new procedure, that the name of the procedure is the word that follows TO, and that you need to be in the Edit mode so you can write the procedure.

Second, a procedure name should describe what the procedure does. When you wrote the procedure to draw a square, you gave it the name SQUARE. But you could just as well have given the procedure the name RECTANGLE or TRIANGLE. Logo allows you to give a procedure any name at

all, even if the name has nothing to do with what the procedure does. But it's easier to remember what a particular procedure does if the name matches the result in some clear-cut way, like SQUARE for a procedure that draws a square.

Finally, Logo requires that procedure names be limited to one word. If you try to use a space within a procedure name, the computer interprets the space as the end of the name and ignores any characters that follow it. For example, the computer would treat the name MY SQUARE as if it were just MY. But there may be situations when you would want to use more than one word in a procedure name to remind you what the procedure does. In such cases, simply type the words one after the other without any spaces between them (as in MYSQUARE) or use periods instead of spaces to separate the words (as in MY.SQUARE). Both forms are acceptable and are commonly employed by people using Logo.

More Things to Do

Some of the drawings in the following activities might extend to the bottom of the screen, so you may want to use the <f5> key to switch to FullScreen. As you may recall from Chapter 3, using the <f5> key allows you to see the bottom parts of these drawings that would otherwise be hidden by the five lines of text at the bottom.

Write a procedure to draw a rectangle with a length of 100 steps and a height of 50 steps. Give it the name RECTANGLE. Then, write a procedure to draw a triangle with all three sides 50 steps long and give it the name TRIANGLE. Finally, copy the following procedure, called CIRCLE, which draws a circle with a diameter approximately equal to 60 steps.

```
TO CIRCLE
 REPEAT 36 [FD 10 RT 10]
END
```

The figure drawn by the CIRCLE procedure isn't really a smooth circle; it's a 36-sided figure that looks like a circle because all of its sides are relatively short. This is a standard way of producing an approximately circular figure in Logo. Of course, you could obtain an even closer approximation to a circle by using a figure with 360 sides, as you did in Chapter 3. For example, replacing the command REPEAT 36 [FD 10 RT 10] by REPEAT 360 [FD 1 RT 1] in CIRCLE would produce a 360-sided figure that would resemble a circle more closely than using a 36-sided figure. But the difference in appearance on the screen is minor when viewed with the naked eye and it takes much longer for the turtle to draw a 360-sided figure than it does to draw a 36-sided figure. Because the difference is negligible, leave the CIRCLE procedure as it is and accept the 36-sided figure it draws as a reasonably good approximation of a circle.

Viewing Your Procedures

By now, you have a number of procedures in your computer. If you want to see the names of all of the procedures currently in your computer's memory, put the screen in the NoDraw mode and type PRINTOUT TITLES to have the titles of all of the procedures printed out on the screen. An abbreviation for this command, and one that may be easier to remember and use, is POTS (for PrintOut TitleS).

You need to be in the NoDraw mode to see the procedure titles printed out, because if you're in the Draw mode, most of the text will be hidden behind the graphics portion of the screen. If you are already in the NoDraw mode and have writing on the screen, it might be a good idea to clear the screen using NODRAW (or ND) before printing out the procedure titles so that there is less clutter on the screen and the titles are easier to read.

Printing Your Procedures

You can also have the computer print out any particular procedure you want to look over in the NoDraw mode by using the command PRINTOUT (or PO) followed by a space and the name of that procedure. The entire procedure appears on the screen. For example, to view the entire SQUARE procedure, type:

```
PRINTOUT SQUARE
```

or, you can type the shorter version:

```
PO SQUARE
```

Try it now just to see how it works.

Using the PRINTOUT command allows you to look at a procedure but not change it. There's no chance of accidentally changing it when you don't want to. If you print out a procedure and discover in looking through it that you want to make changes after all, use the EDIT command to display it in the Edit mode and then make the changes you want.

When you've finished looking at a procedure that's displayed using PRINTOUT, use NODRAW to clear the screen.

If you type PRINTOUT (or PO) without the name of a procedure following it, the computer assumes you want it to print out the last procedure you wrote, edited, or printed out.

You can also print out more than one procedure at a time on the screen, or all of the procedures currently in the computer. To print out more than one, use PRINTOUT, followed by the names of all of the procedures you want to see. Put a set of brackets around the names and spaces between the names. For example, the following command prints out the SQUARE, TRIANGLE, and CIRCLE procedures:

```
PRINTOUT [SQUARE TRIANGLE CIRCLE]
```

If you want to print out all of the procedures currently in the computer, use the command PRINTOUT ALL. The procedures appear on the screen one at a time and, as new ones

appear at the bottom of the screen, the old ones move up to the top of the screen and disappear. If you want to stop the display at any point, press <CTRL> W. Press any key on the keyboard when you want to continue. If you don't want to continue, press <CTRL> G to abort the command.

Erasing a Procedure

If there is a procedure on the list that you decide you don't need any more, you can erase it from the computer's memory using the command ERASE (or ER) followed by a space and the name of the procedure. For example, if you no longer wanted the SQUARE procedure in the computer, you'd simply type ERASE SQUARE (or ER SQUARE) and SQUARE would be removed from the computer's memory. You could verify that SQUARE was no longer in memory by printing out all of the current procedure names using POTS and noting that SQUARE was no longer on the list.

You'll need all of the procedures you've written so far for the next section, but if you didn't, you could also erase two or more procedures at the same time by enclosing the procedure names within brackets, as in ERASE [SQUARE TRIANGLE CIRCLE]. Or, you could erase all of the procedures currently in the computer at one time with the command ERASE ALL.

Saving Procedures and Screen Pictures

Now that you know how to list the procedure names and the actual procedures that are currently in the computer, you're ready to learn how to save, or store, these procedures and screen drawings on a disk. Before doing this, use POTS to make sure all the procedures you want to save are in the computer's memory and you don't have any extraneous procedures that you don't want to waste disk space on.

If there are any procedures listed that you don't want to save, such as any of the procedures from your Activity Disk

which are already saved on that disk, simply erase them using the ERASE (or ER) command. Then, look at the procedures to see if there are any that aren't listed that you would like to save on disk for easy access later, such as the SQUARE procedure you could have erased in the previous section. If so, write them now and then use POTS once again to make sure they're all included in the computer's memory. Now you're ready to learn how to initialize a blank disk and use it to save your procedures and a screen drawing you'll make shortly.

Initializing a Blank Disk

A disk is like audio or video tape. When you put a disk in your disk drive, you can save the information that's currently in the computer's memory for use later. You will then be able to use the information even after the computer has been turned off and then on again. You can load, or transfer, this stored information back into the computer's memory from the disk as often as you want, without having to type it in again on the keyboard.

If you don't have any blank disks, go to your local computer store and buy a box of ten. (Blank disks are generally not sold individually.) The Commodore 64 system uses 5¼-inch, single-sided floppy disks, so that's the kind you should ask for. Any brand will do, since they are interchangeable among microcomputers.

When you buy a box of disks from a computer store, each disk in the box starts out completely blank. Unlike audio or video tape, before you can use the disk, you need to initialize it so that it works with your particular computer system. Initializing, sometimes called formatting, lays down markers on a blank disk so that the computer can keep track of the information saved on the disk and can quickly find it later.

Once a disk is initialized for your computer system, you can save information in many different computer languages on that same disk, as long as all the work is done on the same

brand of system. For example, you'll be able to save information from your Commodore 64 computer in both the BASIC and Logo languages on the disk you'll initialize in this section. But, usually, you can't use a disk that's initialized for one brand of system to transfer information with another brand. For example, the disk you'll initialize for use with your Commodore 64 system will not save or load information into an Apple IIe system.

If you initialize a disk that already has information on it, all the previous information is lost and the newly initialized disk starts out blank.

Initializing a blank disk is very easy when using Commodore 64 Logo, since Commodore 64 Logo has a built-in command that automatically initializes a disk.

First, open the door of your disk drive, place the new disk in the drive with the label up and nearest you, and close the door. Then type:

```
DOS [N0:WORK DISK,23]
```

and press <RETURN>. Don't worry about what this command means. The computer understands it and that's what's important. The red light on the disk drive then goes on and the disk drive runs for approximately two minutes. When the red light goes off and the question mark and cursor reappear on the screen, the disk is initialized. Take it out, put one of the labels that comes in the box of blank disks on it, write "Work Disk" on the label with a felt-tip pen (use a felt-tip pen so you don't harm the disk), and put the disk back in its envelope. Be careful to touch the disk only on the label and not on any exposed parts of the disk itself. If you touch an exposed part of the disk, the disk may be damaged and you may not be able to use it again.

The words WORK DISK will be the first words to appear on your computer's screen when you tell the computer to display all the information you've saved on the disk in the next

section. You can use any phrase that doesn't exceed 16 characters in place of WORK DISK. Some people prefer using their own names so anyone else using the disk can see who initialized it. However, I'll refer to the disk on which you save your procedures and screen drawings as the Work Disk.

About Workspace and Files

When you're working in Logo, the amount of space in the computer's memory occupied by your procedures is called your workspace. Logo allows you to save an entire workspace at one time, but it doesn't allow you to save part of a workspace. That's why you had to erase any procedures you didn't want to save by using the ERASE (or ER) command.

When your current workspace is saved on a disk, it is saved as one complete set of procedures and is referred to as a file. You need to give every file a name so that, when you want to read a file from the disk back into your computer, you can tell the computer the name of the particular file you want it to look for. You'll learn how to assign file names later. The two restrictions on a procedure file name are that it can't contain any spaces and it can't have any more than 11 characters (letters, numbers, or symbols). If a file name has a space in it, the computer ignores any characters that come after the space. Similarly, the computer ignores any characters that occur after the first 11.

The name of a file does not have to have anything to do with the procedures that make up the file. However, just as it's a good idea to call a procedure by a name that indicates what the procedure does, it's also a good idea to call a file by a name that indicates what the procedures in the file do. For example, the file for the activity in Chapter 5 had the file name FIND.ME. The file has several procedures in it that are part of the activity. In particular, the main procedure in the file is a procedure with the name FIND.ME. I specifically used the name of the main procedure for the name of the file so I could easily remember what the file did. I could have

used any other name for the file, but another name probably wouldn't have helped me remember what the file did as well as FIND.ME does.

Saving Your First File

You're now ready to save the procedures currently in your workspace as a file on an initialized disk. So, take out the disk you initialized in the previous section (as I mentioned in that section, I'll refer to this disk as your Work Disk), place it in the disk drive, and close the drive door.

The first thing you need to do is decide on a name for your file. You can use any name. How about FIRST.FILE as a name that's reasonably easy to remember?

To save your workspace, type SAVE, a space, a quotation mark, and FIRST.FILE, the name of the file. Type:

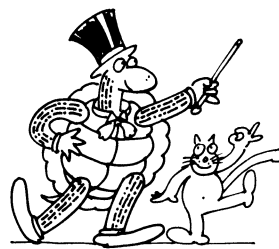
```
SAVE "FIRST.FILE
```

The quotation mark tells the computer that the name to follow is the name of a file, as opposed to the name of a procedure. No quotation mark is needed at the end of the file name.

When you enter this command, the red light on the disk drive goes on and the computer saves all of the procedures in your workspace on the disk in a file with this name. If you forget to put a disk in the drive, the computer will display a message to that effect on the screen. When the red light on the disk drive goes off and the cursor reappears on the screen, you'll know that the workspace is saved.

Saving Screen Pictures

In addition to procedures, you can save screen pictures on a disk so that you can later transfer them from the disk back onto the display screen. But you must have the picture on the screen when you save it. The command for saving a



Turtle Tip: If you have a printer, you may find it convenient to have paper copies of the procedure listings for reference and for easy viewing while you're modifying them. It's also useful and fun to print out paper copies of screen pictures that you or your children have created.

Commodore 64 Logo allows you to do both. You can print out procedure listings using a built-in, primitive command, and you can also print out screen pictures using a special file on the Utilities Disk that comes with your Logo package.

Both the text-printing command and the picture-printing file are designed to be used with the VIC-1525 serial printer—the printer sold by Commodore specifically for use with the Commodore 64 system. If you have any other printer connected to your computer, these commands and this file may not work. If they don't, see a Commodore dealer about possible modifications that will allow you to print out text and pictures on your printer.

The command for printing text on your printer is PRINTER. PRINTER tells the computer to not only display text on the screen but to print it out on the printer. Use the NOPRINTER command to tell the computer to display text only on the screen.

For example, suppose you want to print the procedure listing for SQUARE on the printer. You do this by typing PRINTER; then PRINTOUT (or PO), a space, and SQUARE, the name of the procedure. Turn off the printer by typing NOPRINTER. With the abbreviated form of PRINTOUT, this sequence of commands looks as follows:

```
PRINTER
PO SQUARE
NOPRINTER
```

You can also print out more than one procedure listing at a time by typing PRINTOUT (or PO) followed by the names of all the procedures you want to have printed. The procedure names must be separated by spaces and enclosed within a pair of brackets. For example, the commands:

```
PRINTER
PRINTOUT [SQUARE TRIANGLE CIRCLE]
NOPRINTER
```

would tell the computer to print out the procedure listings for SQUARE, TRIANGLE, and CIRCLE.

Similarly, you can print the procedure listings for all the procedures currently in your workspace with the command PRINTOUT ALL (or PO ALL). Using the short form of PRINTOUT, this sequence looks as follows:

```
PRINTER
PO ALL
NOPRINTER
```

To print a screen picture on your printer, you first have to save the picture on a disk using the SAVEPICT command, just as you did with the SQUARES picture earlier in this chapter. When you're ready to print out a copy of the picture on your printer, you use a file called PRINTPICT on the Logo Utilities Disk. PRINTPICT is written in the BASIC language, the language that's built into your Commodore 64 computer. So, when you want to print a picture, you first have to turn your computer off and on again to make sure it's using BASIC rather than Logo.

After you've turned the computer off and on again, place the Utilities Disk in the disk drive, close the drive door, type:

```
LOAD "PRINTPICT",8
```

and press <RETURN>. When the blinking cursor reappears on the screen simply type:

```
RUN
```

and again press <RETURN>.

The red light on the disk drive goes on, indicating that the PRINTPICT file is being transferred from the Utilities Disk to the computer's memory. After a few seconds, the red light goes off and a message on the screen asks for the name of the picture you want to print. Remove the Utilities Disk from the disk drive and replace it with the disk on which the picture you want to have printed is saved. Then, type the name of the picture you want to print. The computer will read the necessary information from the picture file and print out the picture on your printer.

For example, suppose you want to make a copy of the SQUARES picture that you saved on your Work Disk earlier in this chapter. After turning the computer off and on, placing the Utilities Disk in the disk drive, and typing LOAD "PRINTPICT",8 and RUN, you would remove the Utilities Disk from the disk drive and replace it with your Work Disk since the Work Disk contains the SQUARES picture file. Now, simply type the name of the picture file, SQUARES, press <RETURN>, and a copy of that picture will be printed on the printer.

picture is SAVEPICT, space, a quotation mark, and the name you want the picture to have.

Keep in mind that the computer makes a distinction between a procedure that draws a picture and the picture itself. If you want to save the procedure for drawing a picture, you must save it as a file containing the procedure. If you want to save the picture, either by itself or in addition to the procedure, you must first put it on your screen and then save it as a picture. If you write a procedure that draws a picture, saving the picture does not save the procedure.

To see how saving a screen picture works, let's use SQUARE to draw eight squares, with each square rotated 45 degrees from the previous square. Make the drawing and call it SQUARES when you type the SAVEPICT command. Now, type SAVEPICT, a space, a quotation mark, and SQUARES, the name of the picture. The computer saves everything on the screen.

When the computer saves a screen picture on disk, it sees each horizontal line on the display screen as a large number of small rectangles, or pixels, arranged in a row. As a result, everything on the screen is saved as the color of each individual pixel. But when you tell the computer to read a picture back onto the screen, all the pixels in a particular row are read together. So, it looks as if the computer is saving and reading the screen an entire horizontal line at a time.

Asking for a Catalog

Just to make sure your picture and workspace are saved, let's ask the computer to print out a catalog of all the files and pictures on the Work Disk. Type CATALOG. When you press <RETURN>, the disk drive starts working and, after a few seconds, your screen looks as shown in Figure 6-6.

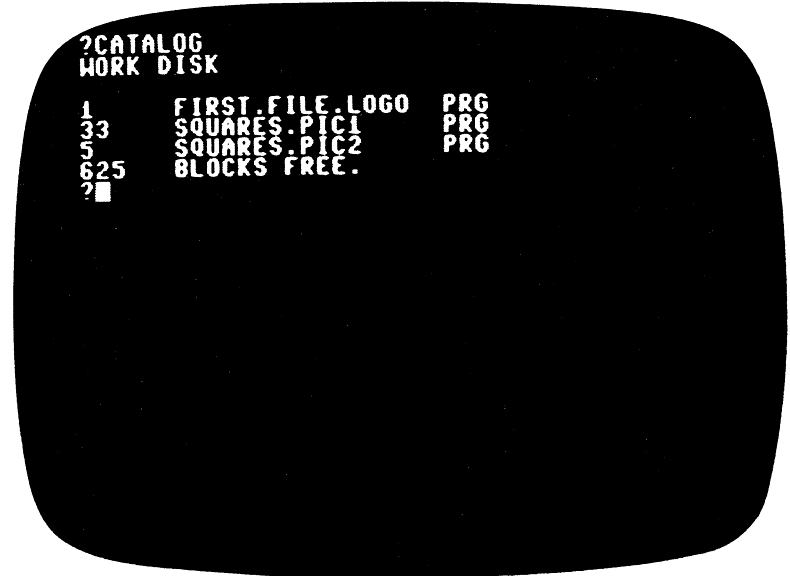


Figure 6-6. The screen as it appears after using CATALOG *to view a catalog of the Work Disk*

The message WORK DISK appears at the top of the screen. (Remember, this is part of the initialization message you typed in when you initialized the disk.) Below the message is a list of all the files currently on the disk, each preceded by a number representing the amount of disk space used by that file and immediately followed by the name of the language in which the file was written (since information written in different languages on the same computer can be saved on the same disk). Finally, the statement BLOCKS FREE appears, preceded by a number representing the amount of empty space still remaining on the disk. Right now, the only files you've saved are FIRST.FILE and SQUARES, so those are the only file names that you should see on the screen. Notice that there are actually two files with the name SQUARES. This is because Logo saves the picture itself in one file and the color information about it in a second file. Together, these two files contain all the information there is about the picture that's been saved. The letters PIC1 and PIC2 appear after SQUARES.

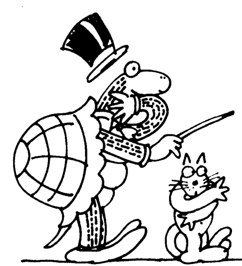
As you save more files on your Work Disk, the amount of free space remaining on the disk decreases. You can use the amount of space listed for each of the files in the catalog to estimate when there is so little space left that you cannot put anything else on the disk. If you forget to check the amount of space remaining and try to save a file that requires more space than is available on the disk, the computer displays a message to that effect on the screen and allows you to replace your current Work Disk with another one. If you don't happen to have another initialized disk handy, you can initialize a new blank disk at that time and then save your workspace on it.

Clearing the Computer's Memory

Saving your workspace on a disk does not erase the procedures from your computer's memory. All of the procedures making up FIRST.FILE are still in your workspace as well as on your Work Disk. They will remain in your workspace until you turn the computer off, erase them individually using ERASE, erase them collectively using ERASE ALL, or type GOODBYE.

When you type GOODBYE, the screen returns to the way it looks when you first start using Logo. The built-in command GOODBYE leaves Logo in the computer but erases everything except the language's built-in, primitive commands. You could also clear the computer's memory by turning the computer off and then on again. But turning the computer off causes the computer to forget Logo and requires you to load Logo in again from the language disk.

This is an important difference between procedures and primitive commands: When you turn the computer off or type GOODBYE, the commands you've defined in procedures are forgotten. Go ahead and clear the computer's memory by typing GOODBYE.



Turtle Trap: When you save a new file on a disk using the same name as a file that's already on that disk, the old file vanishes and the new one takes its place. To make sure that doesn't happen by accident, always get a catalog of the file names on the disk you're using before saving a new file so that you'll know what names you've already used.

On the other hand, there may be times when you deliberately want to replace one file with another having the same name. For example, suppose that tomorrow you load Logo into the computer and develop a couple of very nice designs that you'd like to include as part of FIRST.FILE. You can add them to FIRST.FILE by leaving these new procedures in your workspace, reading the original version of FIRST.FILE into your workspace from the disk, and then going through the saving process to include the enlarged set of procedures under the name FIRST.FILE. The enlarged set of procedures will replace the old set that had the same name. Similarly, you can read two or more files into your workspace one at a time (as long as the workspace can accommodate them), and save them as one file using a new file name.

Loading a File from Disk

The procedures in FIRST.FILE aren't in the computer's memory any more; they're on your Work Disk. You need to transfer them back into the computer's memory when you want to use them. You can read FIRST.FILE back into the workspace by typing:

```
READ "FIRST.FILE
```

When you press <RETURN>, the disk drive starts working and, after a few seconds, the names of the procedures in the file appear on the screen as the computer reads them into its memory. When the drive stops working and the cursor reappears on the screen, the procedures are back in the workspace and ready to be used once again.

The command for reading screen pictures from disk is similar. For example, to read the picture of the squares back onto the screen, type:

```
READPICT "SQUARES
```

The picture is drawn, one line at a time from the top to the bottom, on the display screen.

You can read a file—screen pictures or procedures—into the workspace even if you already have procedures there, as long as there is enough space in the computer's memory. If there isn't enough space in the computer's memory, the screen displays a message to that effect and you'll know you must erase at least some of the procedures currently in your workspace in order to allow the computer to read the new file into its memory.

Erasing Files

If you decide you have no further need for a file and you don't want to take up disk space with it, you can erase it from the disk using the command ERASEFILE, a space, a quotation mark, and then the name of the file. (There is no abbreviated form for the ERASEFILE command.) For example, if you had no further need for FIRST.FILE, you could type:

```
ERASEFILE "FIRST.FILE
```

to erase it from the disk.

Caution: Don't use ERASEFILE unless you have the procedures in that file in your current workspace or you don't want to keep them. Erased files disappear and you can't retrieve them.

Likewise, if you have no further need for a screen picture saved on your Work Disk, you can erase it from the disk. The command for erasing a picture from the disk is ERASEPICT, a space, a quotation mark, and the name of the picture. For example, to erase the picture saved under the name SQUARES, you'd type ERASEPICT "SQUARES.

Logo Activity Time

The activity SPELL'NDRAW on your Activity Disk is designed to make it easy to move the turtle, draw several simple geometric shapes and clear the screen, sending the turtle back to its Home position at the center of the screen. SPELL'NDRAW provides you with a set of ready-made procedures that perform these actions, using procedure names that are simply the first letters of the actions themselves. For example, to draw a square, just type S. To draw a triangle, type T.

All of the drawings are yellow on a blue background, and they are all in the FullScreen format. Experiment with these procedures to draw figures on different parts of the screen and to combine different shapes in the same drawing. If you want to, you can also define additional procedures of your own that draw other geometric shapes. Or, you can change the background or pen colors or give the turtle more ways of moving around the screen. (In the spirit of the game, try to use one-letter procedure names that somehow indicate what the procedure does.)

In order to get SPELL'NDRAW ready for use, place your Activity Disk in the disk drive, close the drive door, type:

```
READ "SPELL'NDRAW
```

and press <RETURN>. When the disk drive stops working, remove the disk.

The procedures and what they do are listed in Figure 6-7. Use them just as you've been using the primitive commands and the procedures in this chapter. For example, you can use the procedures contained in SPELL'NDRAW to create the picture of a house with the sun shining in the sky above it. Use the procedures F, B, R, and L to position the turtle, and the procedure S to draw a square for the frame of the house, T to draw a triangle for the roof of the house, and C to draw a circle for the sun.

Figure 6-7. Procedures in SPELL'NDRAW

Procedure	*Function*
S	Draws a square.
T	Draws a triangle.
C	Draws a circle.
H	Draws a hexagon.
O	Draws an octagon.
P	Draws a parallelogram.
F	Raises the turtle's pen, moves it 10 steps forward, and then lowers the pen.
B	Raises the turtle's pen, moves it 10 steps backward, and then lowers the pen.
R	Turns the turtle 15 degrees to the right.
L	Turns the turtle 15 degrees to the left.
E	Erases the screen. The same as typing DRAW, but puts the display screen in FullScreen format instead of in SplitScreen format.
?	Displays a list of these commands.

Giving procedures one-letter names to make them easier to use, as is done in the activity SPELL'NDRAW, also lends itself to educational applications. The use of one-letter procedure names can help young children develop their spelling skills. Children often are taught to sound out a word and try to guess its first letter when they are learning to spell. By using the first letter of the name of the shape a procedure draws as the name of the procedure, children get practice in saying the word, guessing its first letter, and using the first letter to draw the corresponding picture. If they don't obtain the picture they want, they know they didn't get the first letter of the word right.

I developed the SPELL'NDRAW activity for my almost-five-year-old son Timothy one day when he wanted to draw pictures on my Commodore 64 computer using Logo the way I do. Timothy was learning to sound out words and trying to spell them, so I wrote a set of procedures that each perform one simple action. I named each procedure with the first letter of the action it performed.

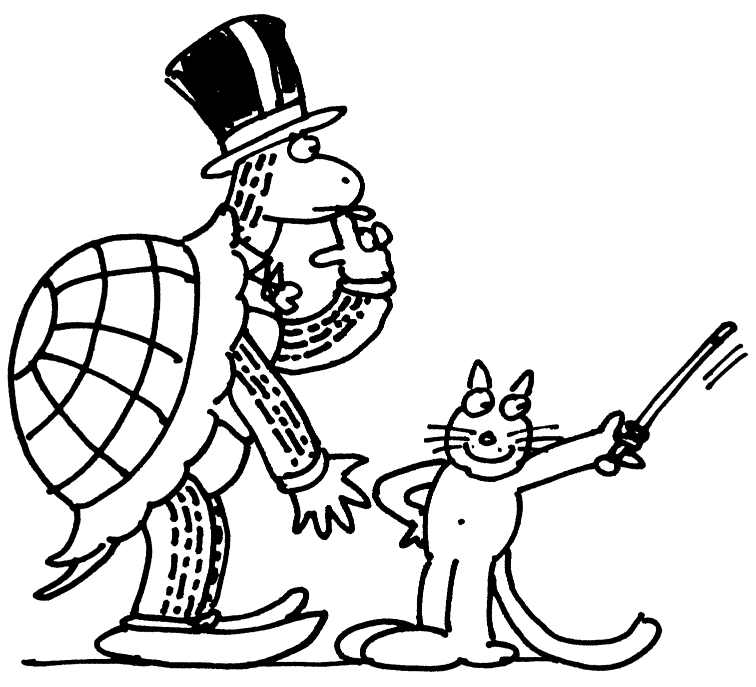
Timothy wanted SPELL'NDRAW to draw several simple geometric figures. After some discussion, we decided to have SPELL'NDRAW turn the turtle to the right or left a little bit at a time, move the turtle with its pen up a short distance forward or backward and then lower its pen, and erase the screen and send the turtle back to its home position. I also made sure that every procedure included a blue background (blue is Timothy's favorite color), a yellow pen color (yellow is one of my favorite colors), and the FullScreen format so there would be no distracting text on the screen to block the view of the turtle or its drawings.

SPELL'NDRAW not only provides practice in sounding out the first letters of words—the first step in learning to spell them—it also makes it much easier for a young child to remember and type long procedure names correctly, allowing the child to use Logo and enjoy the creative and intellectual benefits it has to offer. Using one-letter procedure names is also widely employed for people who are physically handicapped and find it difficult to press keys. For them, the fewer keys that have to be pressed, the better.

New Commands Introduced In This Chapter

Command	*Abbreviation*
TO	
END	
GOODBYE	
ERASE ALL	
EDIT	ED
ERASE	ER
PRINTOUT	PO
PRINTOUT TITLES	POTS
CATALOG	
SAVE	
SAVEPICT	
READ	
READPICT	
ERASEFILE	
ERASEPICT	
<CTRL> C	

7 A Closer Look at Procedures



Using Logo's built-in, primitive commands to create procedures (as you did in Chapter 6) is only the beginning. Once you create them, Logo lets you use the procedures to define new ones. You can also modify old procedures to make them do something new. And you can stop procedures at any point to look at their results, make changes, or move on to something else.

Using Procedures Within Procedures

As we discussed previously, once you define a procedure you can use it as if it's a primitive Logo command. As a matter of fact, you can use it as a procedure within another procedure—as what is called a subprocedure. Let's look at an example.

Let's write a procedure with a subprocedure that draws a square in the upper right corner of the screen with X, Y coordinates (50,50) instead of with the center of the screen as the starting point. To do this, you need to lift the turtle's pen, move the turtle to the appropriate position with SETXY 50 50, put its pen down, and have it draw a square. You could have it draw a square by typing the commands you used in defining SQUARE. But it's quicker just to type SQUARE, since you have already told the computer how to draw a square when you defined the SQUARE procedure. Using SQUARE as part of a sequence of commands for drawing a square at (50,50) gives the display shown in Figure 7-1 and is typed as follows:

```
PU
SETXY 50 50
PD
SQUARE
```

In fact, you can make these commands into a new procedure that will do all of this when you type one word. You can name this new procedure SQUARE.50.50 to remind you that it is intended to draw a square starting at the point with X,Y coordinates (50,50). You are going to incorporate the SQUARE procedure into this new procedure, so begin by having the computer read SQUARE into its memory from your Work Disk. SQUARE is in FIRST.FILE, so have the computer read FIRST.FILE into its memory. Make sure that your Work Disk is in the disk drive, then type READ "FIRST.FILE. The red light on the disk drive goes on, indicating the drive is

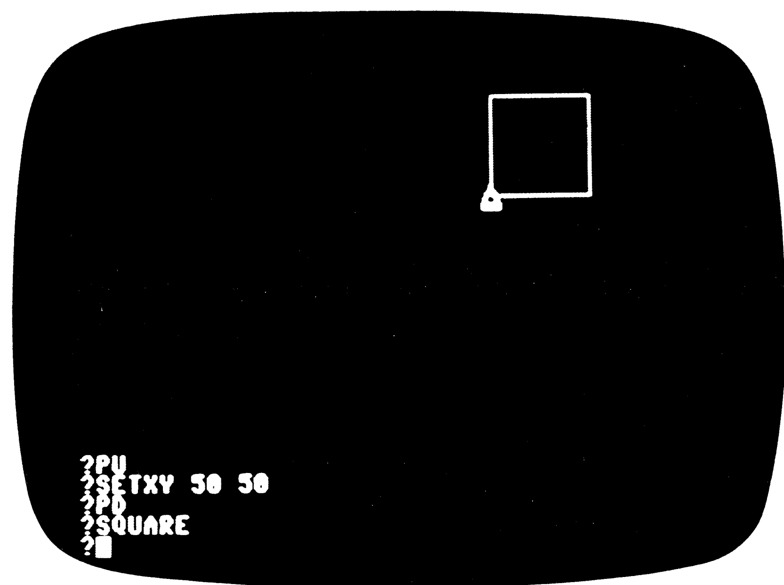


Figure 7-1. A square drawn using SETXY 50 50

working. When the red light goes off, FIRST.FILE, with SQUARE in it, is in the computer's memory ready for you to use.

Now, let's create SQUARE.50.50. Begin by typing:

```
TO SQUARE.50.50
```

to enter the Edit mode. Then, type the commands as they were just given and type END to complete the procedure. The procedure looks as follows:

```
TO SQUARE.50.50
 PU
 SETXY 50 50
 PD
 SQUARE
END
```

Press <CTRL> C and wait for the message:

```
SQUARE.50.50 DEFINED
```

Now you can use this command any time you are in the Draw mode. Simply type:

```
SQUARE.50.50
```

and, wherever the turtle happens to be, it immediately goes to screen location (50, 50) and draws a square with sides 50 steps long.

When a procedure is used within another procedure, the main procedure is called a Level 1 procedure and the subprocedure is called a Level 2 procedure. For example, since SQUARE is a subprocedure within SQUARE.50.50, SQUARE.-50.50 is Level 1 and SQUARE is Level 2. If SQUARE had a subprocedure, it would be Level 3.

Level numbers come in handy when you try to run a procedure and can't because there's a problem somewhere in it. When that happens, the screen displays an error message telling you what the problem is and at what level you can find it. The error message tells you:

- The nature of the error (for example, was a bracket missing in a REPEAT statement, or was a procedure name used for which no procedure was defined?).
- The line in the procedure in which the error occurred.
- The name of the procedure in which the error occurred.
- The level of the procedure or subprocedure in which the error occurred.

Stopping a Procedure

There may be times when you'll want to stop a procedure while it's running—maybe you want to look at the result or make changes, or maybe you want to move on to something else. At those times, three commands you might find useful are <CTRL> G, <CTRL> W, and <CTRL> Z.

The command <CTRL> G stops, or "aborts," your procedure after it's started, in case you don't want to wait until the action is completed to move on to something else. For example, if you are running the SQUARE.50.50 procedure and press <CTRL> G, the procedure immediately stops at whatever stage of execution it happens to be. You can then type DRAW to clear the drawing screen and go on to other Logo work.

The command <CTRL> W makes a procedure pause after it's started so you can inspect the result at that stage of its development at your leisure. For example, suppose you're using a procedure that draws a design composed of 12 triangles and you want to see what the design looks like after only six of the triangles are drawn. All you do is wait until six triangles are drawn and then press <CTRL> W to freeze it at that point. You unfreeze the procedure by pressing any key on the keyboard or <CTRL> W, but not <CTRL> G. Pressing <CTRL> G, of course, permanently stops, or aborts, it.

The command <CTRL> Z, like <CTRL> W, causes a procedure to pause when you give the command. But a procedure interrupted by <CTRL> Z, unlike one interrupted by <CTRL> W, can only be resumed by typing the word CONTINUE or the abbreviation CO. And, while it is frozen, you can type in any commands you want before making it resume its operation.

For example, suppose you are drawing 12 triangles in red on a black background. Let's say you decide that you'd like to see what the design looks like with the first six triangles in red and the last six triangles in green. You'd start the procedure and, when the first six triangles had been completed, you'd press <CTRL> Z to freeze the procedure at that point. Then, you'd change the pen color from red to green by typing PENCOLOR 5 (or PC 5), and then type CONTINUE (or CO) to tell the computer to resume drawing the design, with the pen color green instead of red. You can't use the command <CTRL> W to change pen colors while a

procedure is running, because as soon as you start to type PENCOLOR 5 by pressing the P key, the procedure resumes its execution. In essence, <CTRL> W is used just to pause and look, while <CTRL> Z is used to pause and possibly use some commands before resuming the procedure.

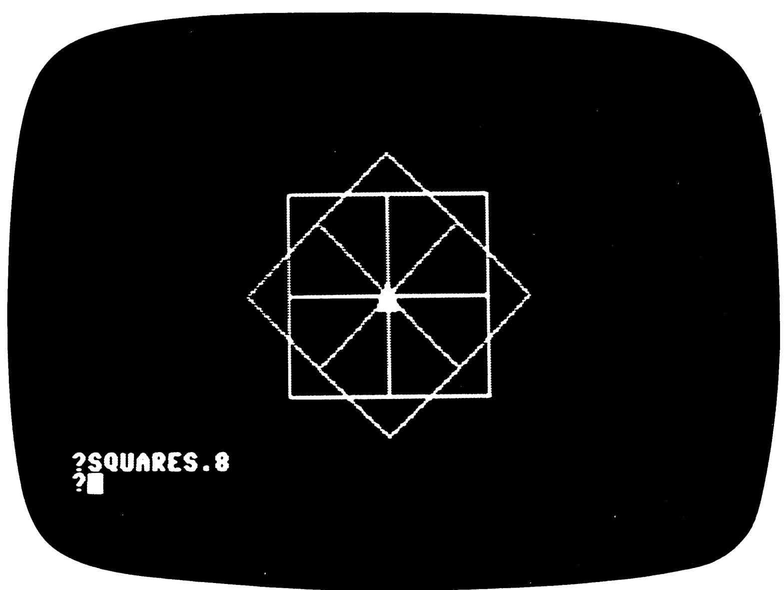
Try practicing with these three commands when running some of your procedures to become familiar with what each does and how each differs from the others.

More Things to Do

In Chapter 5, you drew several designs composed of squares, triangles, and other simple figures. Now, using procedures within procedures, you can draw designs that repeat those simple figures. For example, one design was composed of eight squares, each rotated 45 degrees to the right of the previous one, as shown in Figure 7-2. The following simple procedure, called SQUARES.8, uses the SQUARE procedure to draw this same design:

```
TO SQUARES.8
 REPEAT 8 [SQUARE RT 45]
END
```

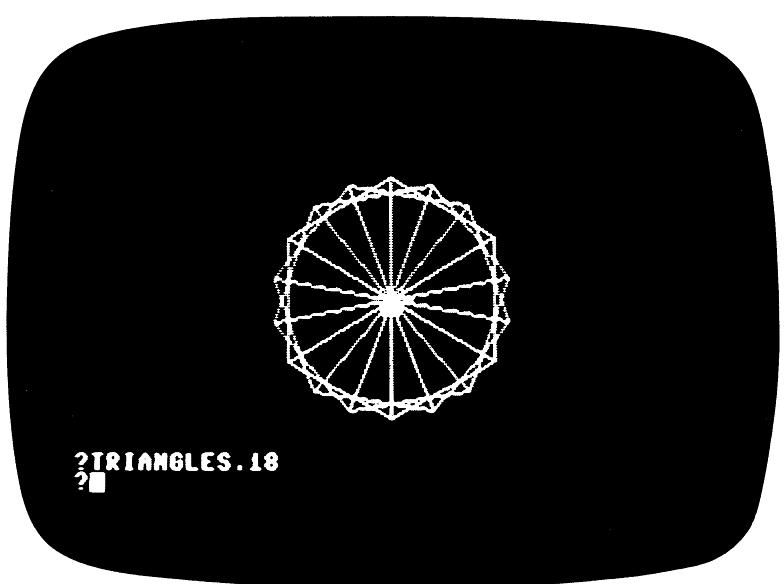
Figure 7-2. The design drawn by SQUARES.8



Enter the SQUARES.8 procedure and test it out.

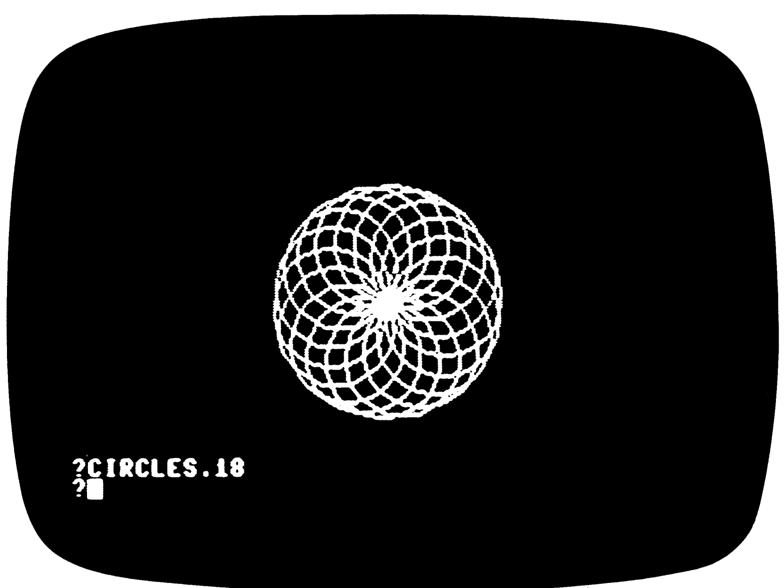
Now, use the RECTANGLE, TRIANGLE, and CIRCLE procedures to develop procedures similar to SQUARES.8. Try different numbers of repetitions and different degrees of rotation to see how the design changes. For example, you might define the procedure SQUARES.12 containing 12 repetitions of a square with a rotation of 30 degrees between each one. Using a 30-degree rotation with 12 repetitions produces a total rotation of 360 degrees ($12 \times 30 = 360$) or one complete revolution. For example, TRIANGLES.18 and CIRCLES.18 (18 repetitions with an angle of rotation of 20 degrees) give the designs illustrated in Figures 7-3 and 7-4.

Figure 7-3. The design drawn by TRIANGLES.18



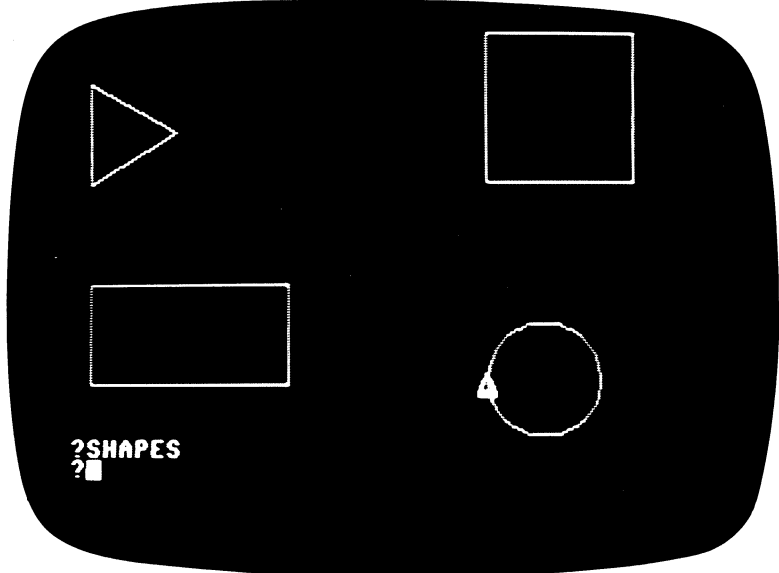
Once you've found some designs you like, try drawing them on different colored backgrounds and with different colored pens. A red, yellow, green, or blue pen drawing on a black background is striking and gives a three-dimensional effect. Keep track of the design and color combinations you like best so that you can save them in a file on your Work Disk.

Figure 7-4. The design drawn by CIRCLES.18



Now, try to use SQUARE, RECTANGLE, TRIANGLE, and CIRCLE to create a procedure called SHAPES that draws a triangle in the upper left corner of the screen, a square in the upper right corner, a rectangle in the lower left corner, and a circle in the lower right corner. SHAPES produces a display like the one shown in Figure 7-5.

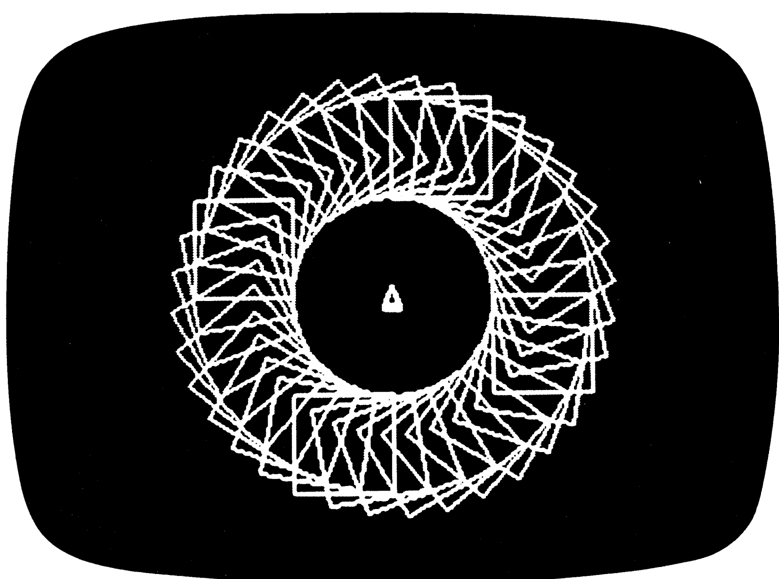
Figure 7-5. The shapes drawn by SHAPES



One final design you might want to create is a "doughnut" composed of repetitions of a simple geometric figure with a "hole" in the middle at the center of the screen. Use the same approach as you did with the earlier designs (such as SQUARES.8, TRIANGLES.12, and CIRCLES.18), but before drawing the figures, move the turtle with its pen up a specified distance from the center. For example, the following procedure, called DOUGHNUT.SQUARES.36, uses 36 repetitions of SQUARE with a rotation of 10 degrees between each drawing, and starts each drawing 50 steps from the center of the screen as shown in Figure 7-6. It also uses a green pen color, just for variety.

```
TO DOUGHNUT.SQUARES.36
 PC 5
 REPEAT 36 [PU FD 50 PD SQUARE PU BK 50 PD RT 10]
END
```

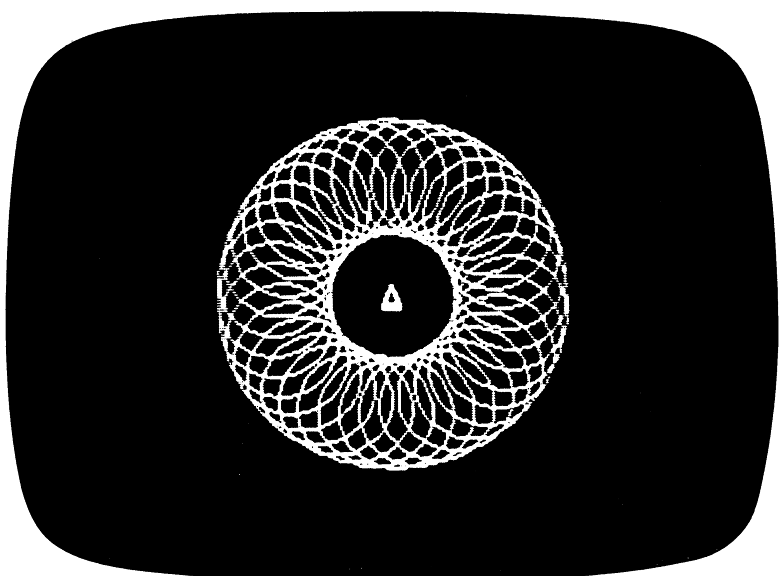
Figure 7-6. The design drawn by DOUGHNUT. SQUARES.36



The design produced by DOUGHNUT.SQUARES.36 is rather large, so, in order to see it all, you need to press the <f5> key to put the screen in the FullScreen format. You can press <f5> before you type the command, after the drawing is completed, or even while the drawing is in progress. Press <f3> to return to the SplitScreen format.

Now try using CIRCLE instead of SQUARE to obtain DOUGHNUT.CIRCLES.36. The resulting design is illustrated in Figure 7-7 and looks very attractive in red on a black background. To obtain these colors, use the pen color and background color commands PC 5 and BG 0 as part of the procedure, after the line containing the procedure name.

Figure 7-7. The design drawn by DOUGHNUT. CIRCLES.36



You can also draw the designs different distances from the center of the screen, and you can make the squares and circles larger or smaller in order to see what effect these

changes have on the design that's produced. See which combination of shapes, colors, distance, and size gives the most pleasing results.

Modifying Procedures in the Edit Mode

Logo procedures aren't written in stone—they can be, and often are, modified in the Edit mode.

Let's suppose that you want to modify the procedure called SQUARE to draw a square with sides 75 steps long. Recall from Chapter 6 that SQUARE draws a square with sides 50 steps long and is given as follows:

```
TO SQUARE
 REPEAT 4 [FD 50 RT 90]
END
```

Just as you did in Chapter 6 when you wanted to make corrections in the Edit mode, you tell the computer you want to modify SQUARE in the Edit mode by typing the word EDIT (or its abbreviation ED), a space, and the name of the procedure, SQUARE. Type:

```
EDIT SQUARE
```

or, you can type:

```
ED SQUARE
```

The computer immediately switches to the Edit mode, with the SQUARE procedure on the screen, as in Figure 7-8.

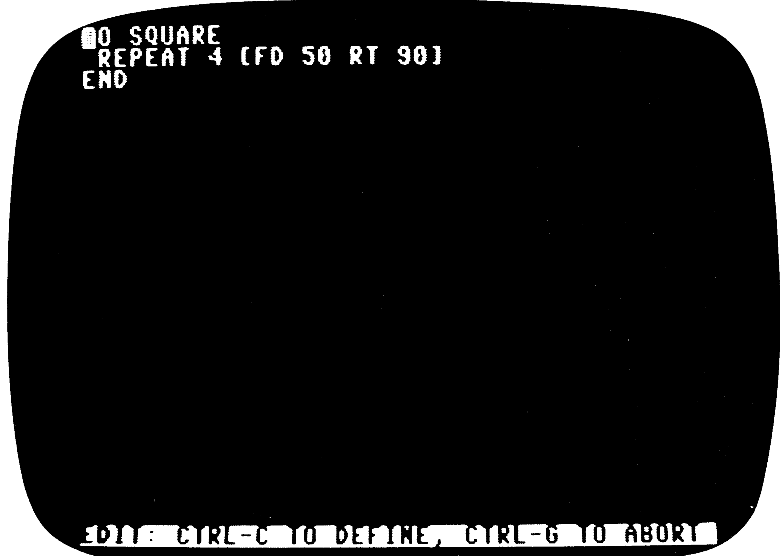


Figure 7-8. The screen as it appears with SQUARE in the Edit mode

Now, you can use the <CRSR> keys and the <DEL> key to make any changes you want in the procedure, as described in the last chapter. To change the length of the sides from 50 to 75, use the <CRSR> keys to position the cursor in the space to the right of the number 50, as in Figure 7-9.

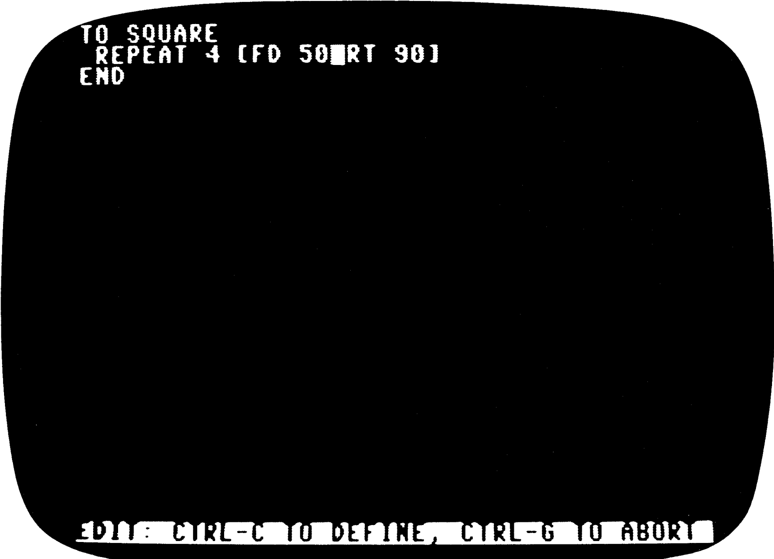


Figure 7-9. Positioning the cursor to the right of characters to be changed (50)

Then, press the <DEL> key twice to delete the characters 5 and 0, as in Figure 7-10. Finally, press the keys 7 and 5 to insert the number 75 where 50 had been, as in Figure 7-11.

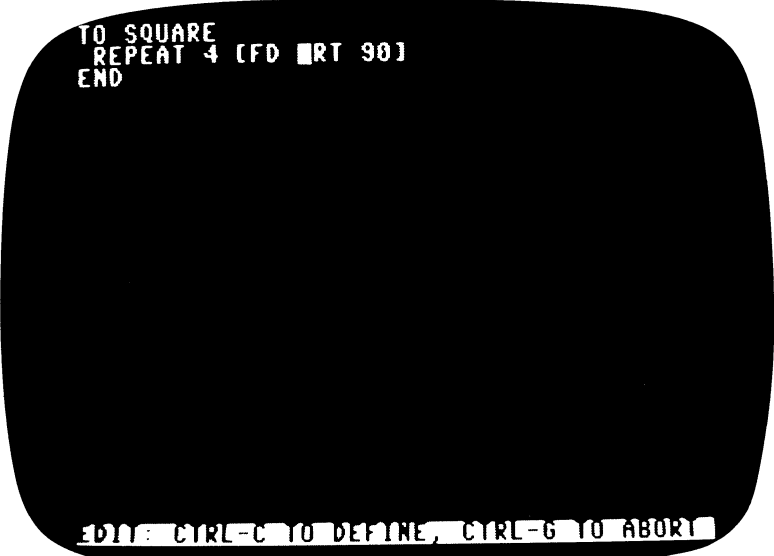


Figure 7-10. Deleting the characters to be changed (50)

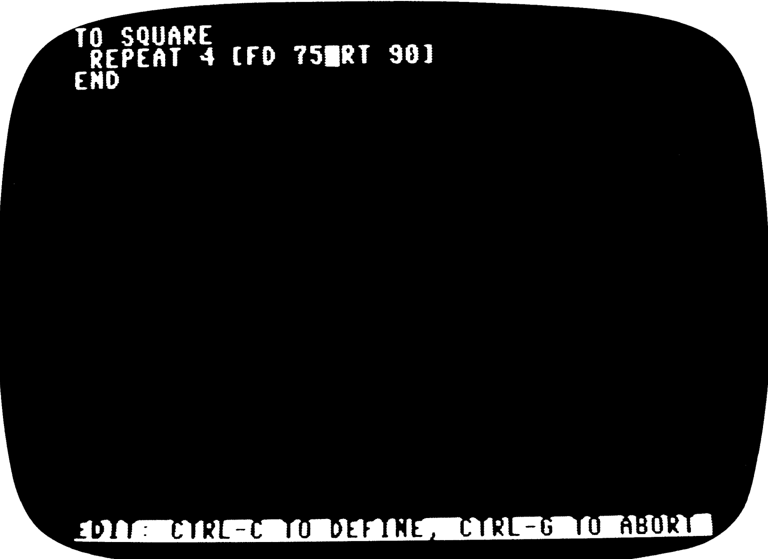


Figure 7-11. Inserting the new characters (75)

When you finish modifying SQUARE, press <CTRL> C to tell the computer you're done. The screen switches to the NoDraw mode and, after a few seconds, the message SQUARE DEFINED appears on the screen with the cursor blinking on and off to the right of a question mark on the line below the message. Your changes have been incorporated into SQUARE. Now when you type SQUARE, the turtle draws a square with sides 75 steps long instead of 50 steps long.

When you finish modifying the procedure, you don't need to move the cursor to a particular position on the screen before pressing <CTRL> C. The same is true when you write a procedure. The cursor can be in any position on the screen in the Edit mode when you press <CTRL> C, telling the computer you are finished.

If you've typed some changes, but decide you'd rather keep the original version of the procedure, press <CTRL> G instead of <CTRL> C. As the message at the bottom of the screen tells you, <CTRL> G aborts, or stops, modification of the procedure, returning you to the NoDraw mode and keeping the procedure as it was first typed.

A quick way to get the last procedure you edited or wrote back on your screen is to type the word EDIT (or ED) without any procedure name following it. When you do, you are automatically put in the Edit mode with the last procedure you defined or modified showing on the screen. Try it now. Type EDIT and the screen switches to the Edit mode, with SQUARE on the screen.

If you want to look at all of the procedures currently in the computer's memory at one time, type EDIT ALL (or ED ALL). Then, you can use the <CRSR> keys to make changes in any of your procedures.

There may be times when you might have so many procedures in the computer's memory that they can't all appear on the screen at the same time. Logo provides several commands that let you move around the screen, and move new material onto the screen, so that you can edit any of the

procedures available. These special editing commands, and their function in the Edit mode, are as follows:

- <HOME> moves the text so the line currently containing the cursor appears in the center of the screen.
- <CTRL> A moves the cursor to the beginning of the line it's currently on.
- <CTRL> B moves the text back one complete screenful, if there's more text than can be displayed on the screen at one time.
- <CTRL> C exits the Edit mode and tells the computer to include the changes that have been made in the definition of the procedure or procedures.
- <CTRL> D deletes the character the cursor is currently on.
- <CTRL> F moves the cursor forward one screenful, when there's more text than can be displayed on the screen at one time.
- <CTRL> G exits the Edit mode without making any changes, leaving the procedures as they were before you entered the Edit mode.
- <CTRL> K deletes (or "kills") all of the characters to the right of the cursor on the line it's currently on.
- <CTRL> L moves the cursor to the end of the line it's currently on.
- <CTRL> N moves the cursor down one line (to the "next" line).
- <CTRL> O takes everything to the right of the cursor on its current line and moves it down one line. If the cursor is at the beginning of a line, it opens a blank line at that part of the procedure.

- <CTRL> P Moves the cursor up one line (to the "previous" line).

The best way to see what these editing commands do is to try them out. So, type:

```
EDIT ALL
```

to put all of your current procedures into the Edit mode. Then, practice using the editing commands to become familiar with them.

Using Additional Commands Within Procedures

Several commands introduced earlier can be incorporated into the procedures you write, making them more efficient and complete. The following commands can be used within procedures just as you use them outside of procedures:

- BACKGROUND (BG)
- PENCOLOR (PC)
- HIDETURTLE (HT)
- SHOWTURTLE (ST)
- SINGLECOLOR
- DOUBLECOLOR
- TEXTSCREEN
- SPLITSCREEN
- FULLSCREEN

Turtle Trap: When you use the commands SINGLECOLOR or DOUBLECOLOR within a procedure, be careful where in the procedure they occur. As we discussed earlier, using either of these two commands causes the screen to automatically erase any pictures, switch to the SplitScreen format (if it isn't already in that format), and change the background and pen to their default colors of dark gray and white (if they aren't already these colors). Because of this, always put the commands SINGLECOLOR and DOUBLECOLOR before the procedure commands that set the drawing screen format, the background color, and the pen color.

Let's use some of these commands in the procedure DOUGHNUT.SQUARES.36 developed earlier in this chapter. Try making the drawing on the screen in the FullScreen format, with a black background and a green pen color. Hide the turtle until the drawing is complete in order to speed up the action. As the turtle moves around the screen when it's visible, the computer not only has to draw the trail, it also has to continuously redraw the turtle itself. Using the HIDETURTLE command frees the computer from having to redraw the turtle over and over again, speeding up the process.

So that you don't need to spend time looking for the DOUGHNUT.SQUARES.36 procedure, here it is again:

```
TO DOUGHNUT.SQUARES.36
 REPEAT 36 [PU FD 50 PD SQUARE PU BK 50 PD RT 10]
END
```

When you've finished typing it, press <CTRL> C. Now, you can edit it by typing EDIT DOUGHNUT.SQUARES.36.

Use the <CRSR> keys to position the cursor at the end of the first line, just to the right of the name, as in Figure 7-12. Now press the <RETURN> key. This takes you to an empty second line and moves the rest of the commands down one line, as in Figure 7-13. You can now type the commands for

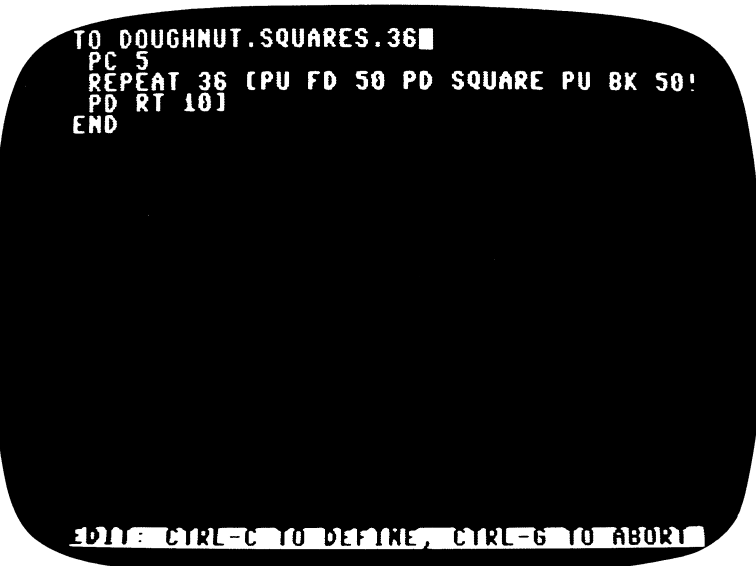


Figure 7-12. Positioning the cursor at the end of the line above where you want to insert a new line

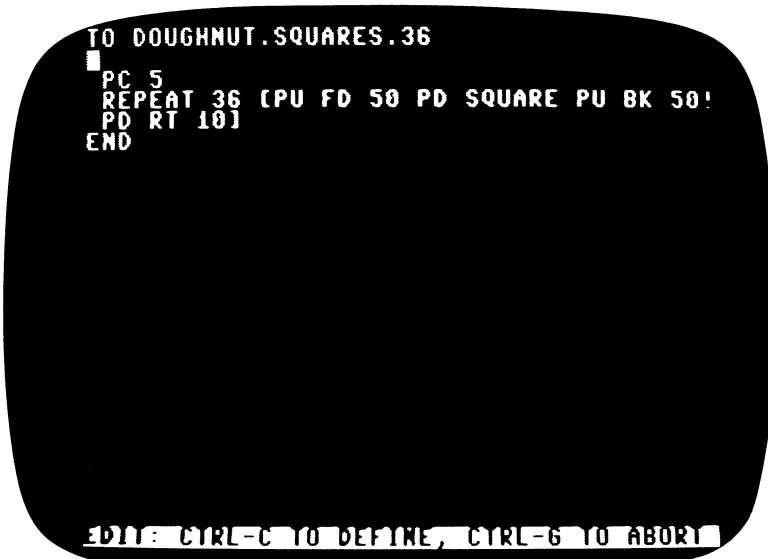


Figure 7-13. Inserting a new line in the Edit mode

the FullScreen format, black background color, and green pen color, either on the same line separated by spaces, or on separate lines by pressing <RETURN> after each command. Using a separate line for each command makes the edited procedure look like this:

```
TO DOUGHNUT.SQUARES.36
 FULLSCREEN
 BG 0
 PC 5
 HT
 REPEAT 36 [PU FD 50 PD SQUARE PU BK 50 PD RT 10]
 ST
END
```

Press <CTRL> C to tell the computer your editing is completed. When the message DOUGHNUT.SQUARES.36 DEFINED appears on the screen, the changes have been implemented. You can type DOUGHNUT.SQUARES.36 to verify that the changes you made have been incorporated into the procedure.

From now on, all graphics procedures in this book will include the line FULLSCREEN BG 0 so that the screen is in the FullScreen format with a black background. Designs are easier to see in the FullScreen format and black is a distinct background for most pen colors. Most of the procedures used from now on will leave the pen color at its default value of white, since white on black is a good combination. But feel free to change both the background and pen colors if you prefer something other than what is used here.

More Things to Do

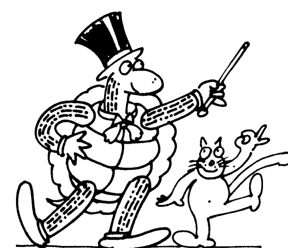
Take some of the procedures you wrote earlier and practice editing them by changing the size of the designs, the background and pen colors, or the screen location where the designs are drawn.

Use the HIDETURTLE command to make some of your designs appear more quickly, but remember to include SHOWTURTLE before the end, so the turtle reappears. For fun, try hiding the turtle while drawing the first half of a design, then showing it while drawing the second half. For example, suppose your design consists of eight squares, with each square turned 45 degrees to the right of the previous one. Your procedure could start with the HIDETURTLE command, followed by a command involving REPEAT and SQUARE to draw the first four squares using an invisible turtle. This could be followed by the SHOWTURTLE command, and then a command involving REPEAT and SQUARE to draw the last four squares using the visible turtle. A procedure that does just this, called HIDE.AND.SHOW, follows:

```
TO HIDE.AND.SHOW
 FULLSCREEN BG 0
 HIDETURTLE
 REPEAT 4 [SQUARE RT 45]
 SHOWTURTLE
 REPEAT 4 [SQUARE RT 45]
END
```

Since HIDE.AND.SHOW uses SQUARE as a subprocedure, make sure that SQUARE is in your computer's memory when you try to run HIDE.AND.SHOW.

Write a procedure that draws several of your designs on the screen, one after the other, with the screen clearing after each drawing is completed. For example, with the turtle hiding and the screen in the FullScreen format, start by drawing a simple yellow square on a blue background. Use DRAW to clear the screen, then draw a triangle, then another DRAW, then a circle. Clear the screen with DRAW, change the background to black and the pen color to red, and draw one at a time SQUARES.12, TRIANGLES.18, and CIRCLES.18, with DRAW between each set of figures. Incorporate all of this in one super procedure called MY.ARTWORK. Try it. It's good practice in developing a fairly complex and sophisticated



Turtle Tip: As you and your child continue through this book, you'll write a large number of procedures that illustrate the capabilities of Commodore 64 Logo. Many are used throughout the chapter in which they're presented, and throughout the rest of the book. In order to avoid having to retype them every time they're called for, you should save the procedures you develop in each chapter as a file on your Work Disk. That way, you can read them back into the computer from the disk. I suggest using the file name CHAPTER.7 for the procedures developed in Chapter 7, CHAPTER.8 for those developed in Chapter 8, and so on.

procedure using previously defined subprocedures as building blocks—just the sort of learning activity that Logo's procedure capability is designed to encourage.

Spend some time practicing the techniques for working with procedures in order to become familiar with their purpose and use. Writing, editing, and saving procedures play a fundamental role in all of the applications of Logo covered later in the book. It will help if you feel comfortable working with procedures before proceeding.

Logo Activity Time

The game TIC.TAC on your Logo Activity Disk is a tic-tac-toe game pitting you against the computer. At the start of the game, the turtle draws a tic-tac-toe grid on your screen while hidden. I hid the turtle with HIDETURTLE to show you how you can draw designs more quickly using an invisible turtle. I also used a yellow pen color on a blue background to show you how you can use different pen colors and background colors to make a design more attractive.

Instead of the Xs and Os used in the traditional game, TIC.TAC uses squares to represent your moves and triangles to represent the computer's moves.

Read the file TIC.TAC into your computer from your Logo Activity Disk by typing:

```
READ "TIC.TAC
```

When the file has been read into your workspace, run the main procedure, called TIC.TAC, by typing:

```
TIC.TAC
```

When you press <RETURN>, your screen switches to the Draw mode with a SplitScreen format. The hidden turtle then draws a tic-tac-toe grid in the graphics part of the screen, with the numbers 1 through 9 in each cell of the grid.

A message at the bottom of the screen asks if you want to go first. If you type YES, the computer displays a second message asking you to type in one of the numbers from 1 to 9 to indicate where in the grid you want your square to be drawn. The hidden turtle then draws a square in the cell of the grid specified by your answer. The computer randomly selects one of the remaining empty cells in the grid and uses the turtle to draw its mark, a triangle, in that cell. If you had answered NO instead of YES when asked if you wanted to go first, the computer would have made its selection first and then asked you to make your selection.

Play alternates between you and the computer. The computer automatically checks to make sure the cell you select is empty before having the turtle draw a square there. If you accidentally select a cell that has already been used, the computer will display a message asking you to make another selection.

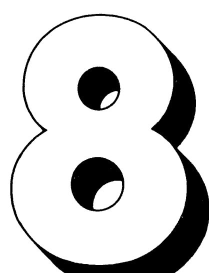
The game ends when either you or the computer position three of your marks in a row horizontally, vertically, or diagonally. The computer displays a message at the bottom of the screen indicating which of you won the game. The message also tells you to type TIC.TAC if you want to play again. If all nine of the cells are filled without either you or the computer obtaining three marks in a row, the computer displays a message at the bottom of the screen indicating that the game was a draw and neither of you won.

If you want to, you can edit the main procedure (TIC.TAC) so that the turtle is visible throughout the game. To make the turtle visible, type EDIT TIC.TAC to place the TIC.TAC procedure in the Edit mode. Then, use the <CRSR> keys to position the cursor to the right of the command HT (HIDETURTLE) and press the <DEL> key twice to delete this command. Finally, press <CTRL> C to leave the Edit mode with this change implemented. When you run TIC.TAC now, the turtle remains visible. Of course, the original version of TIC.TAC on your Logo Activity Disk still contains the HT (HIDETURTLE) command, so the next time you read this file into your workspace and run TIC.TAC, the turtle will be hidden. If you want to keep a permanent copy of the edited version as well, you can use your Work Disk and the commands you learned in Chapter 6 for saving files on a disk. Give your edited version the name TIC.TAC.2 to distinguish it from the original procedure. When you want to play TIC.TAC with a hidden turtle, read the file TIC.TAC into your workspace from your Activity Disk. When you want to play it with the turtle showing, read the file TIC.TAC.2 into your workspace from your Work Disk.

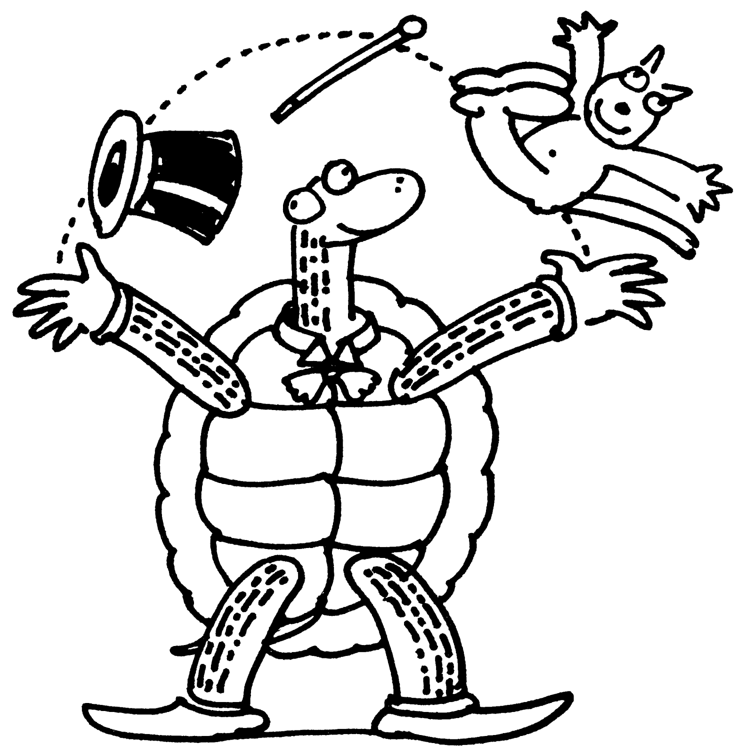
The next two chapters expand on the writing and use of procedures by introducing two powerful programming techniques that can be incorporated into procedures—variables and recursion. Once you feel reasonably comfortable with the basic techniques of procedures, turn to Chapter 8 and begin with the first of these two important topics, variables.

New Commands Introduced In This Chapter

Command	*Abbreviation*
<CTRL> G	
<CTRL> W	
<CTRL> Z	
CONTINUE	CO
<HOME>	
<CTRL> A	
<CTRL> B	
<CTRL> D	
<CTRL> F	
<CTRL> K	
<CTRL> L	
<CTRL> N	
<CTRL> O	
<CTRL> P	



Adding Variety With Variables



Powerful as they may seem to you right now, the procedures used in Chapters 6 and 7 are child's play compared to what you can do with procedures.

In this chapter, for instance, you'll learn how to write a procedure flexible enough to draw squares of many different sizes. You do that by replacing numerical values that don't change (called constants) with some that do (called variables). Using variables, you can

make your square a different size every time you use that procedure.

Before you begin drawing squares, however, let's make sure that your workspace is empty so you can save the procedures you develop in this chapter as a separate file on your Work Disk.

If you've just turned your computer on and loaded Logo into the computer's memory, your workspace is blank and you can begin this chapter's new work immediately. If you have some procedures in the computer, you can clear your workspace by typing GOODBYE.

Now let's get on with our introduction to the use of variables in procedures.

Starting with One Variable

The first procedure you wrote in Chapter 6, called SQUARE, looked like this:

```
TO SQUARE
 FULLSCREEN BG 0
 REPEAT 4 [FD 50 RT 90]
END
```

As you recall, SQUARE produces a square with sides 50 turtle steps long. The commands FD 50 RT 90 within the brackets of the REPEAT statement tell the turtle to take 50 steps forward four times, with a 90-degree turn between each set of 50 steps.

The number 50 in SQUARE is called a constant because it remains the same every time you use the procedure and always produces a square with sides 50 turtle steps long.

Wouldn't it be convenient if you could use a word, like SIDE, in place of the constant 50? Wouldn't it be convenient to be able to simply tell the computer what value you want SIDE to have each time you use SQUARE? As a matter of fact, that's just what Logo allows you to do. And that's how you're able to use one procedure to produce many different results.

The word SIDE, as it's used here, is called a variable name, or just a variable, because the specific value—the number it represents—could vary from one use of SQUARE to the next.

Logo isn't usually very demanding, but it insists that variables start with a colon and contain no spaces. So, in order to comply with Logo's requirements, let's use the variable :SIDE to refer to the lengths of the sides of the squares you draw. Using the variable name :SIDE instead of the constant value 50 in SQUARE makes SQUARE look like this:

```
TO SQUARE :SIDE
 FULLSCREEN BG 0
 REPEAT 4 [FD :SIDE RT 90]
END
```

Notice that :SIDE appears both in the REPEAT statement and in the TO statement. When you write a procedure containing a variable name, Logo also requires you to type the variable name on the first line with the procedure name, leaving a space between the procedure name and the variable name. That way, when you go back to use the procedure, the computer knows it needs to wait for you to type the numerical value you want the variable to have after you've typed the procedure name.

Go ahead and write SQUARE :SIDE now. Remember, to implement the definition (or put it in the computer's memory), press <CTRL> C. When the message SQUARE DEFINED appears on your screen, you can use this new command to draw a square with sides any length you choose. Just be sure to type the length you want the sides to be (the numerical value of the variable) after the name of the procedure. For example, if you want to use the new SQUARE procedure to draw a square with sides 75 steps long, type:

```
SQUARE 75
```

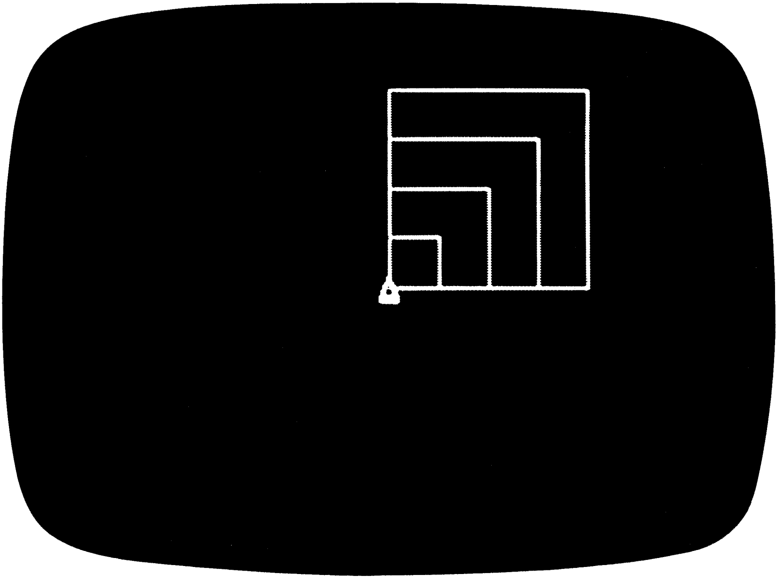
Try it now to verify that it works. If it doesn't, go back into the Edit mode by typing EDIT SQUARE (you don't need to use the variable name to get back into the Edit mode) and try to find and correct any mistakes in the procedure.

Just for fun, use this new SQUARE procedure to draw four squares, one inside the other. Have all four begin at the Home position and make their sides 100, 75, 50, and 25 steps long. You can do all of this with only four commands:

```
SQUARE 100
SQUARE 75
SQUARE 50
SQUARE 25
```

These four commands produce the design in Figure 8-1.

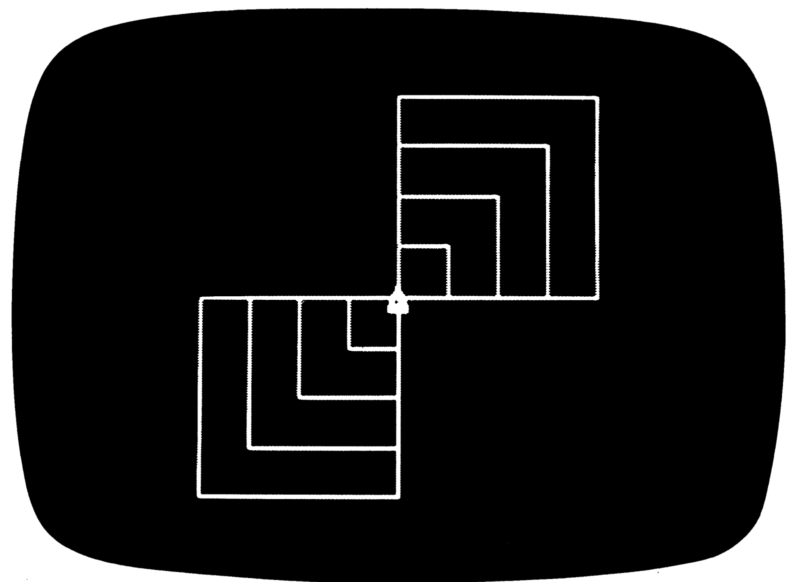
Figure 8-1. Four squares drawn by SQUARE 100, SQUARE 75, SQUARE 50, and SQUARE 25



You can draw four more squares positioned below and to the left of center, mirror images of the first four above and to the right of center as illustrated in Figure 8-2. Without clearing the screen, type these four additional commands:

```
SQUARE -25
SQUARE -50
SQUARE -75
SQUARE -100
```

Figure 8-2. Four squares drawn by SQUARE −25, SQUARE −50, SQUARE −75, and SQUARE −100



Using negative numbers with the FORWARD command makes the turtle move backward instead of forward. So, FD −25 produces the same result as BK 25. The same is true for −50, −75, and −100.

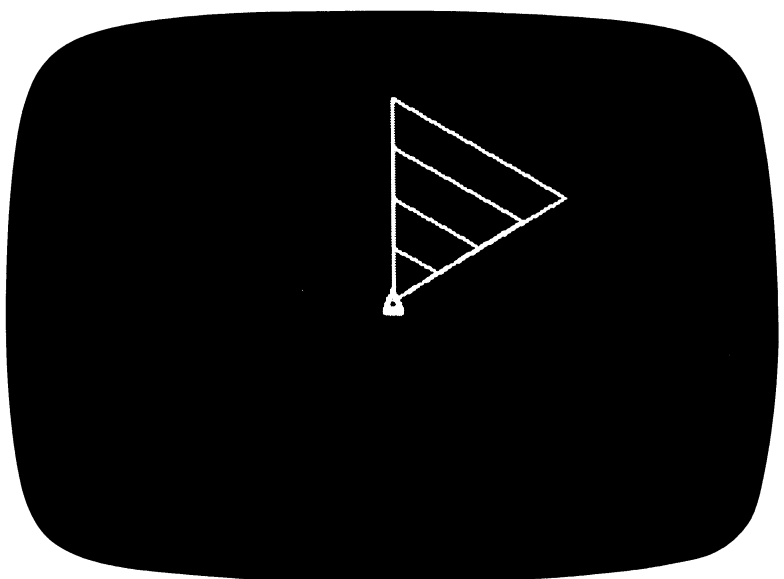
You can use SQUARE as a model to write a similar procedure that draws triangles with sides any length you specify. Call this triangle procedure TRI and use :SIDE again as the variable name for the length of each side. You could call

the procedure TRIANGLE, but TRI is shorter and requires less typing. Here's how the procedure looks:

```
TO TRI :SIDE
 FULLSCREEN BG 0
 REPEAT 3 [FD :SIDE RT 120]
END
```

Go ahead and type this procedure. Now, try it out using 100, 75, 50, and 25 steps for the lengths of the sides, just as you did with SQUARE. The resulting drawing should consist of four triangles, as illustrated in Figure 8-3.

Figure 8-3. Four triangles drawn by TRI 100, TRI 75, TRI 50, and TRI 25



As you may recall, in Chapter 4 you learned a simple but workable process for filling in a square: moving the turtle back and forth across the figure in lines, with a 90-degree turn to the right, a step, and then a 90-degree turn to the left, between each line.

Use the variable SQUARE procedure as a model for writing a corresponding procedure that fills in squares of any length sides. Call this procedure FILL.SQUARE. You'll

have to use a variable name for the length of the sides, just as SQUARE does. To keep things simple, use :SIDE.

To write the FILL.SQUARE procedure, you have to know how far forward and backward the turtle needs to move in each repetition, and how many repetitions are needed to completely fill in the square. Looking at the method used in Chapter 4 for filling in a square with sides 50 steps long, it's clear that both the distance forward and backward and the number of repetitions must be equal to the length of the side of the square, :SIDE. Therefore, the FILL.SQUARE procedure should be:

```
TO FILL.SQUARE :SIDE
 FULLSCREEN BG 0
 REPEAT :SIDE [FD :SIDE BK :SIDE RT 90 FD 1 LT 90]
END
```

Define this square-filling procedure. Then, test it out by drawing a square with sides 60 steps long and filling it in with the consecutive commands:

```
SQUARE 60
FILL.SQUARE 60
```

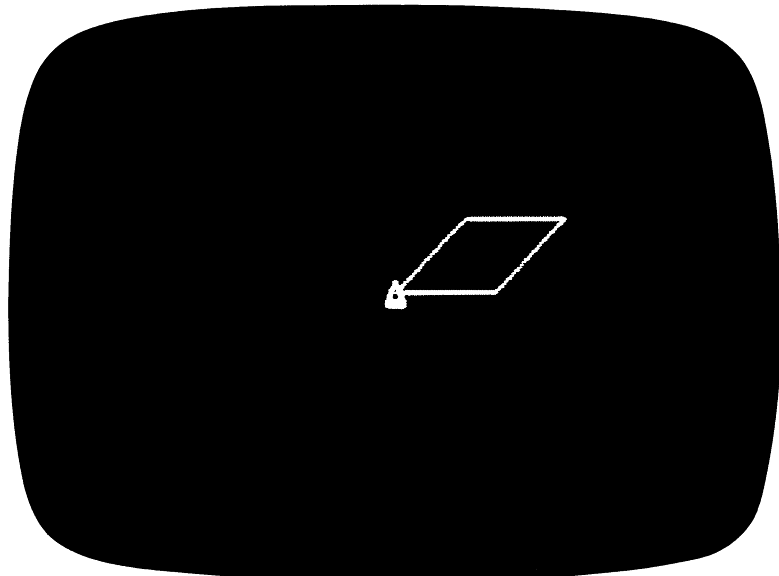
More Things to Do

A rhombus is a geometric figure that looks like a square leaning at an angle. Mathematicians define a rhombus as a four-sided figure with all four sides the same length and with opposite sides parallel to each other. The following procedure draws a rhombus leaning 45 degrees to the right with sides 50 steps long:

```
TO RHOMBUS
 FULLSCREEN BG 0
 RT 45
 REPEAT 2 [FD 50 RT 45 FD 50 RT 135]
 LT 45
END
```

Using the RHOMBUS procedure will produce the drawing shown in Figure 8-4.

Figure 8-4. A rhombus drawn by RHOMBUS



Modify RHOMBUS so that it allows you to draw a rhombus with any length side you specify, instead of always drawing the rhombus with sides 50 steps long. Use the variable name :SIDE to represent the length of the sides, so the first line of the generalized procedure looks like this:

```
TO RHOMBUS :SIDE
```

Once you've written and defined RHOMBUS, test it out by drawing rhombuses with several different length sides.

Now, write a corresponding FILL.RHOMBUS procedure that fills in a rhombus of any length sides the same way you developed FILL.SQUARE to fill squares. Test your rhombus-filling procedure using rhombuses of different length sides, as you did with squares and the square-filling procedures.

For variety in the RHOMBUS and FILL.RHOMBUS procedures, you might want to use a variable name to represent the turtle's pen color number, so you can draw or fill in the figure with a different pen color each time.

Adding More Variables

Once you've written and used a procedure containing one variable, it's really not much harder to write and use a procedure with two or more. All you do is list all of the variable names (separated by single spaces) following the procedure name in the first line. Use a different name for each variable and try to pick variable names that help you remember what each variable stands for.

When you want to use a procedure with several variables, type the procedure name, a space, and the numerical value of each variable you included within the procedure. Like the variable names they represent, the numerical values are separated by single spaces. If you try to use a procedure without providing a number for every variable name, the computer displays a screen message informing you that your procedure needs more inputs.

An input is a piece of information you give the computer to help it carry out a command. For instance, the numbers that the variable names stand for are inputs.

A good example of a procedure with more than one variable name is the following RECT procedure:

```
TO RECT :HEIGHT :LENGTH
 FULLSCREEN BG 0
 REPEAT 2 [FD :HEIGHT RT 90 FD :LENGTH RT 90]
END
```

The RECT procedure draws rectangles with any heights and lengths you choose by using the variable names :HEIGHT and :LENGTH instead of constant values for the height and length.

To draw a short, wide rectangle, as in Figure 8-5, use a number for :HEIGHT that's smaller than the number for :LENGTH, such as 50 for :HEIGHT and 100 for :LENGTH. To draw a tall, narrow rectangle, as in Figure 8-6, use a number for :HEIGHT that's larger than the number for :LENGTH,

Figure 8-5. A short, wide rectangle drawn by RECT

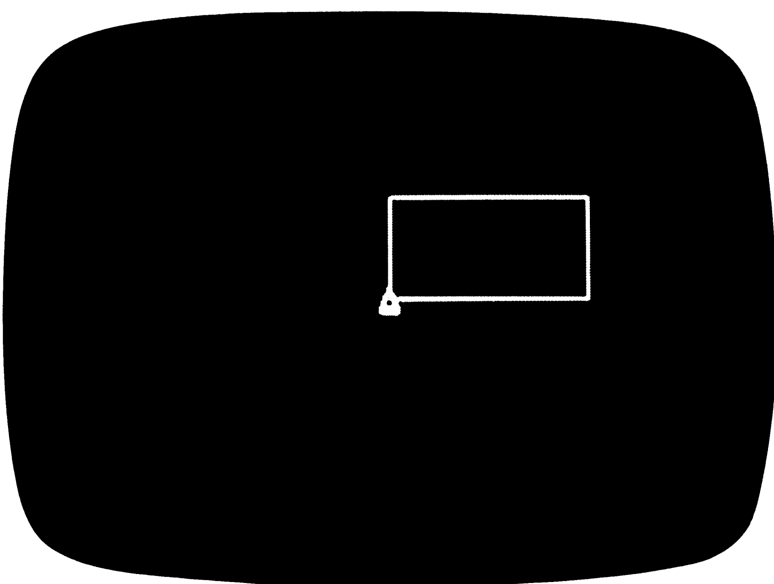
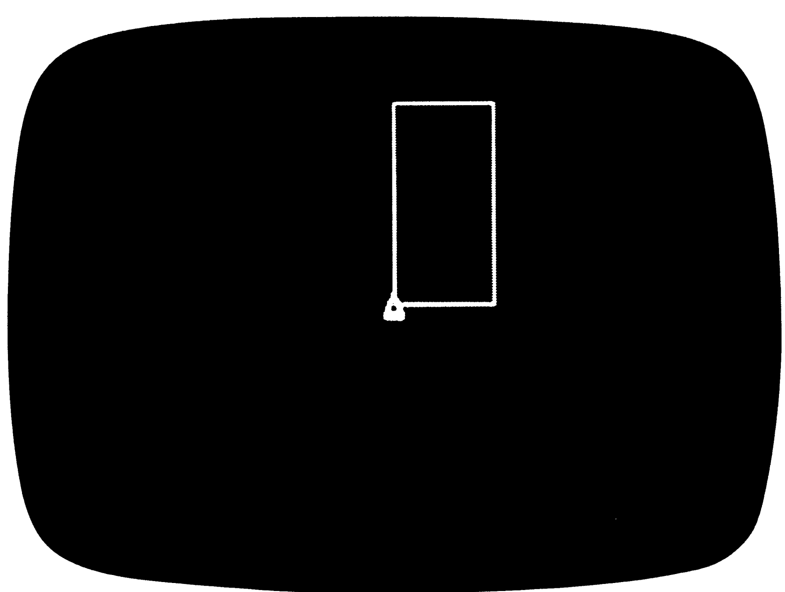
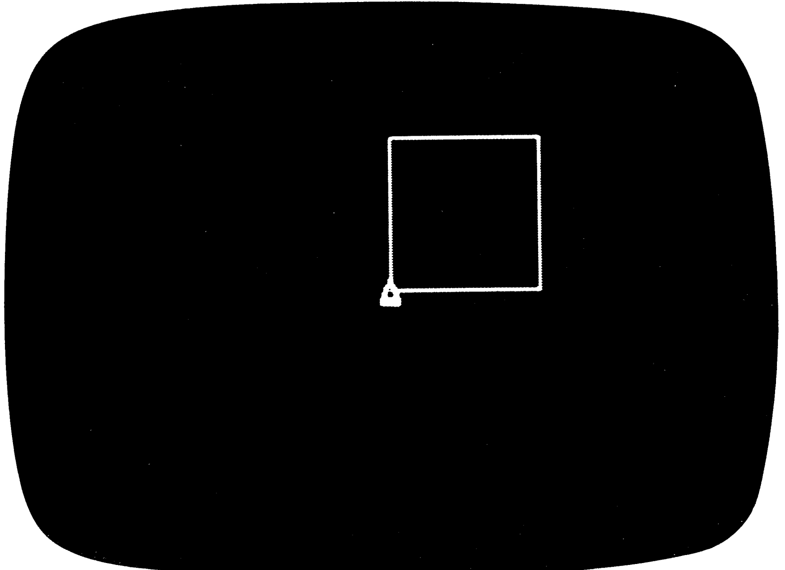


Figure 8-6. A tall, narrow rectangle drawn by RECT



such as 100 for :HEIGHT and 50 for :LENGTH. If you use the same number for both :HEIGHT and :LENGTH, such as 75 for :HEIGHT and 75 for :LENGTH, you get a square as in Figure 8-7.

Figure 8-7. A square drawn by RECT



More Things to Do

In the previous section, I introduced a new figure called a rhombus, which looked like a square leaning at an angle. A rectangle leaning at an angle is called a parallelogram. Mathematicians define a parallelogram as a four-sided figure with opposite sides parallel to each other and of equal length. Figure 8-8 illustrates a parallelogram with a height of 75 steps and a length of 100 steps, leaning 45 degrees to the right.

Write a procedure, called PARA, that allows you to draw parallelograms that lean 45 degrees to the right and have varying heights and lengths. (Use the RHOMBUS procedure as a guide.) Then, try to write procedures that can be used to fill in rectangles and parallelograms of different heights and lengths. Call these procedures FILL.RECT and FILL.PARA.

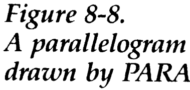
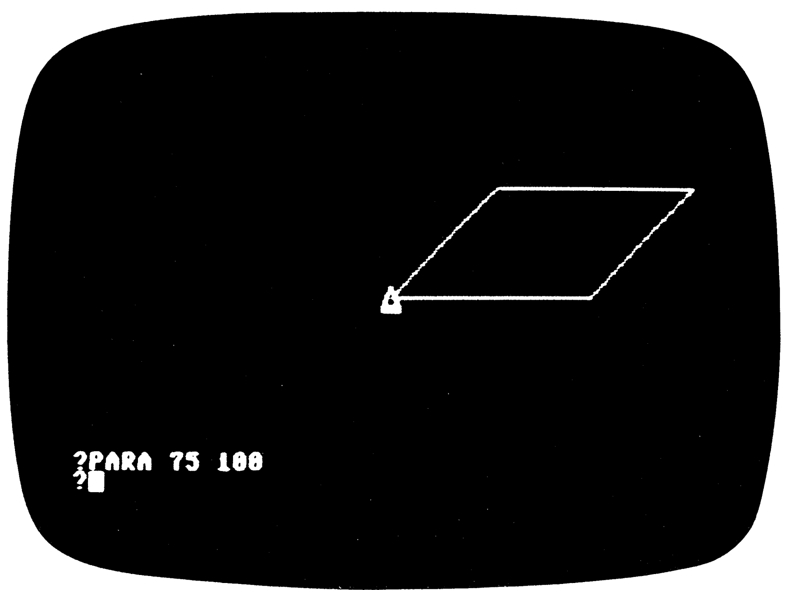


*Figure 8-8.
A parallelogram
drawn by PARA*



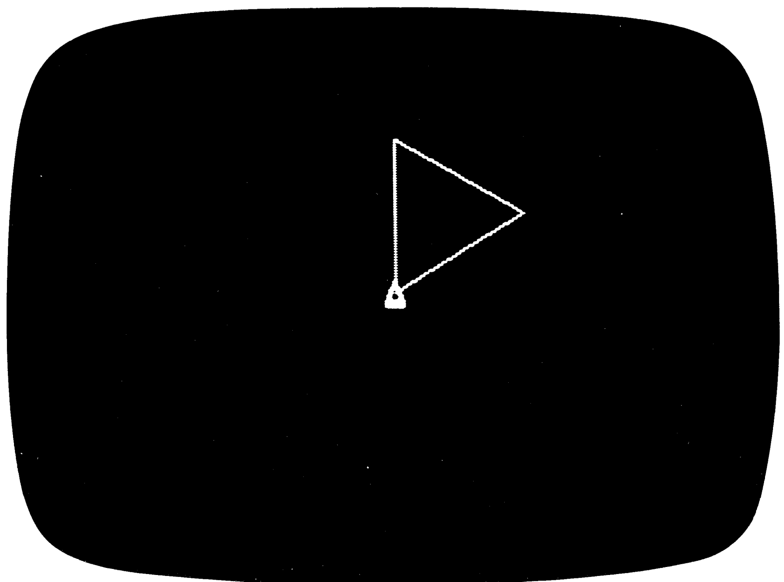
Polygon Patterns

In Chapter 3, you learned how to make the turtle draw polygons, or closed geometric figures, with a particular number of sides. Using procedures containing variable names, you now can write a single procedure that not only draws any of these polygons, but draws them with any length side you want to use. In the following procedure, called POLY, :NUMBER and :SIDE are the names of the two variables:

```
TO POLY :NUMBER :SIDE
 FULLSCREEN BG 0
 REPEAT :NUMBER [FD :SIDE RT 360/:NUMBER]
END
```

The slash (/) stands for division. Most computers use the slash instead of the standard mathematics division sign (÷). The variable called :NUMBER stands for the number of sides you want the polygon to have. The variable called :SIDE stands for the length of each side. For example, POLY 3 75 draws a triangle (a three-sided polygon) with sides 75 steps long, as shown in Figure 8-9; POLY 4 60 draws a square

Figure 8-9. A triangle drawn by POLY 3 75



(a four-sided polygon) with sides 60 steps long, as shown in Figure 8-10; and POLY 5 50 draws a pentagon (a five-sided polygon) with sides 50 steps long, as shown in Figure 8-11.

Figure 8-10. A square drawn by POLY 4 60

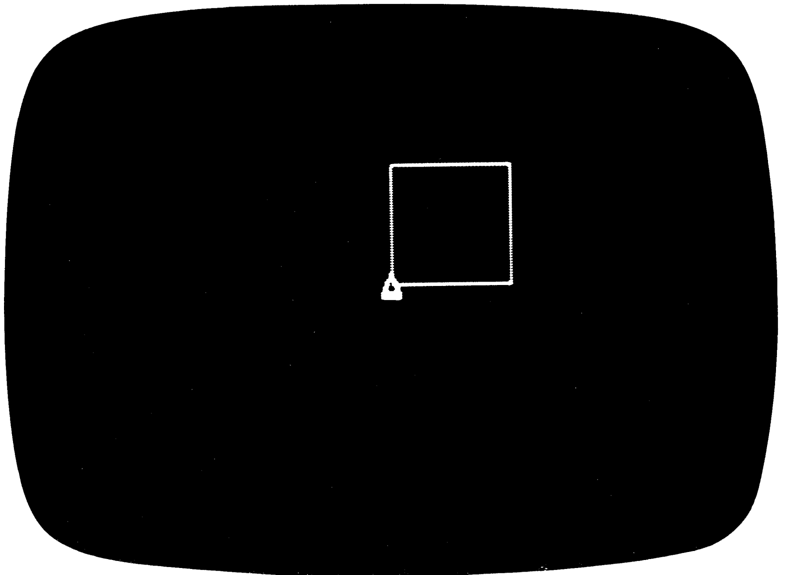
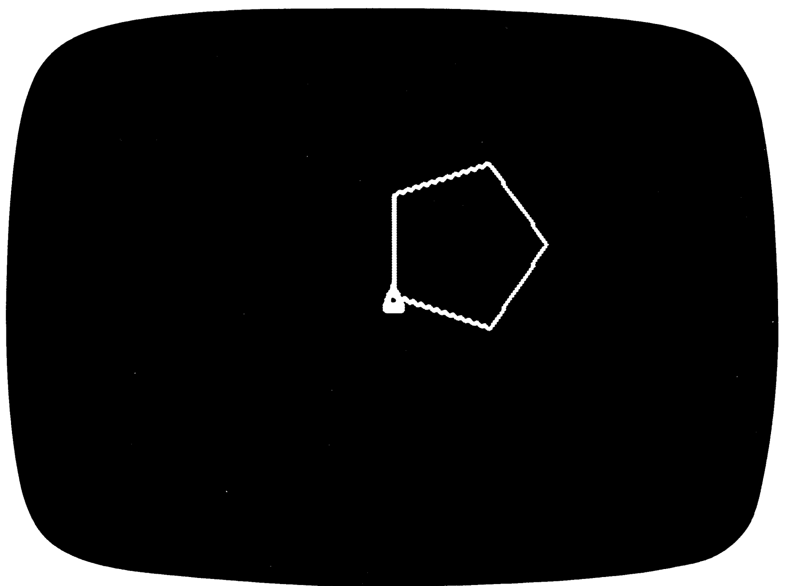
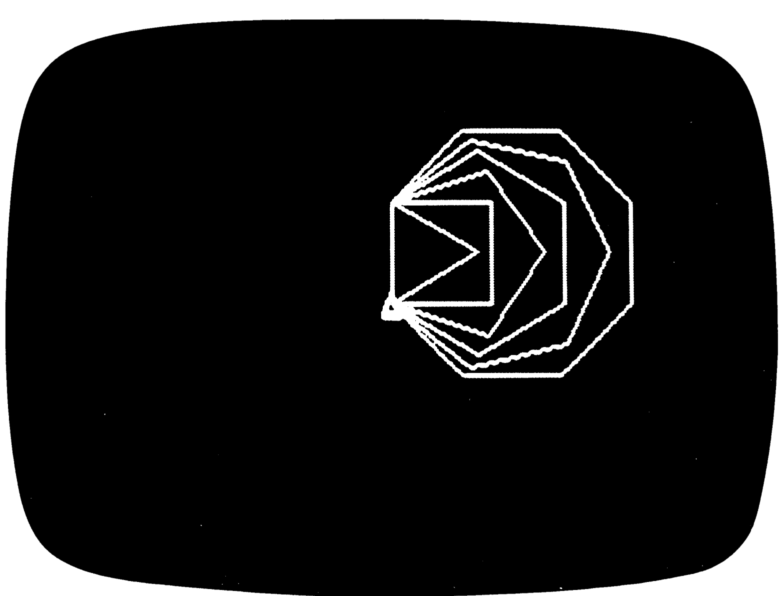


Figure 8-11. A pentagon drawn by POLY 5 50



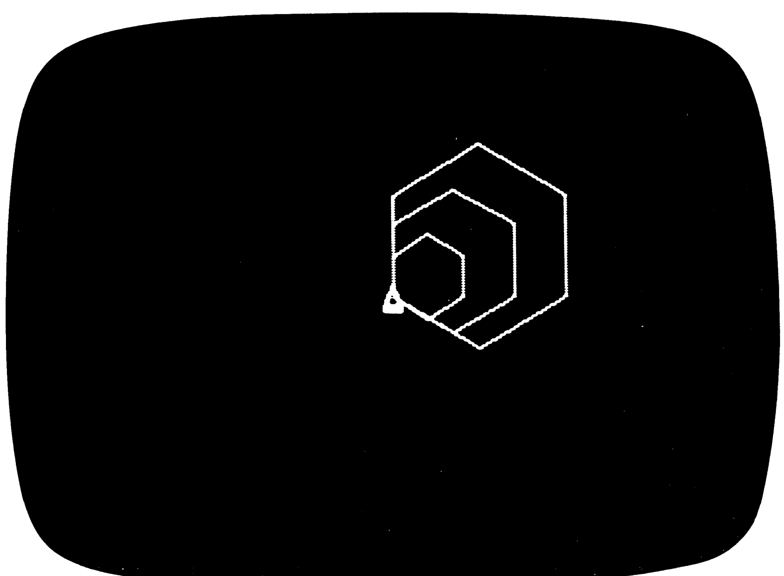
You can use POLY to draw some pretty designs composed of polygons. For example, if you give :SIDE a value of 50 steps and give :NUMBER several different values, such as 3, 4, 5, 6, 7, and 8, you get a drawing like the one pictured in Figure 8-12.

Figure 8-12. A design drawn by POLY, with several values for :NUMBER



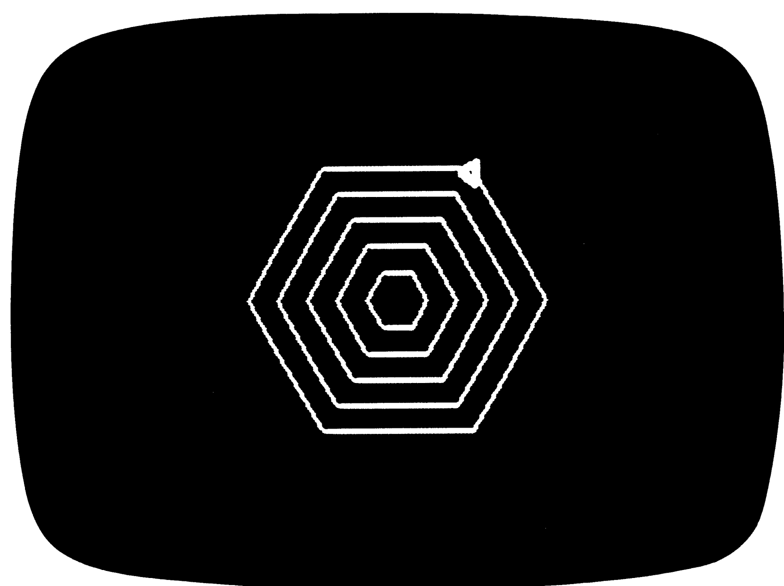
Or, you could give :NUMBER the numerical value 6 and give :SIDE several different values, such as 50, 35, and 20, to get a drawing like the one in Figure 8-13.

Figure 8-13. A design drawn by POLY, with several values for :SIDE



One way to improve on the design in Figure 8-13 is to center each polygon within the next larger one, rather than having all of them start at the same position. For example, suppose you want to draw a design composed of five increasingly larger hexagons, all centered at the turtle's Home position. To do that, start the turtle at the Home position, raise its pen, move it away from Home a selected distance in a straight line, lower its pen, draw a hexagon, raise its pen, move away from Home the selected distance, and draw a new, slightly larger hexagon, continuing the process every 15 steps or so, until all five hexagons are drawn. The design you obtain is shown in Figure 8-14.

Figure 8-14. A hexagon design drawn by HEX.5



You can get the same design using the procedure HEX.5, which uses POLY to draw five hexagons. HEX.5 also uses a procedure called POSITION to raise and lower the turtle's pen and move it 15 steps away from Home each time. In the next chapter, you'll learn how to write procedures like HEX.5 in an even simpler way.

The HEX.5 and POSITION procedures look like this:

```
TO HEX.5
 FULLSCREEN BG 0
 RT 30
 POSITION
 POLY 6 15 LT 120
 POSITION
 POLY 6 30 LT 120
 POSITION
 POLY 6 45 LT 120
 POSITION
 POLY 6 60 LT 120
 POSITION
 POLY 6 75 LT 120
END
```

```
TO POSITION
 FULLSCREEN BG 0
 PU
 FD 15 RT 120
 PD
END
```

Another interesting design you can make with polygons is a wreath, with each figure a given distance away from the center of the screen and tangent to (that is, touching without overlapping) the figure on either side of it. The following procedure, called PENT.12, draws a wreath consisting of 12 tangent pentagons:

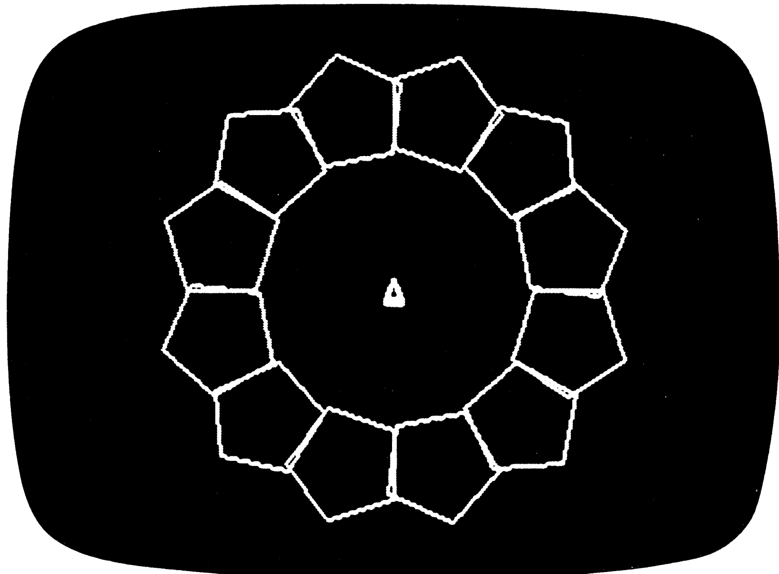
```
TO PENT.12 :DISTANCE
 FULLSCREEN BG 0
 REPEAT 12 [PU FD :DISTANCE PD POLY 5
  :DISTANCE/2 PU BK :DISTANCE RT 30 PD]
END
```

Caution: The REPEAT statement in PENT.12 is too long to fit across the width of this page, so it is broken into two lines. To let you know the second line is a continuation of the first, there's a little less space between them than there normally would be and the second line is indented a little more. From now on, when you see two lines with very little space between them and the second line indented a little more than the first, you'll know the second line is a continuation of the first. Be sure to type both as one continuous line, pressing the <RETURN> key only at the end of the second line.

PENT.12 has a variable name :DISTANCE so that you can specify how far from the center of the screen the wreath is drawn. PENT.12 70, for example, draws a fairly large wreath

with all the pentagons situated 70 steps from the center of the screen, as shown in Figure 8-15.

Figure 8-15. A pentagon design drawn by PENT.12



You can now use PENT.12 with several different values of :DISTANCE in the same picture to create some very interesting and pleasing designs. For example, the following combination produces the design in Figure 8-16:

```
PENT.12 70
PENT.12 40
PENT.12 20
```

And the next combination of procedures produces the design in Figure 8-17:

```
PENT.12 70
PENT.12 50
PENT.12 34
PENT.12 20
```

Figure 8-16. A design drawn by PENT.12, with several values for :DISTANCE

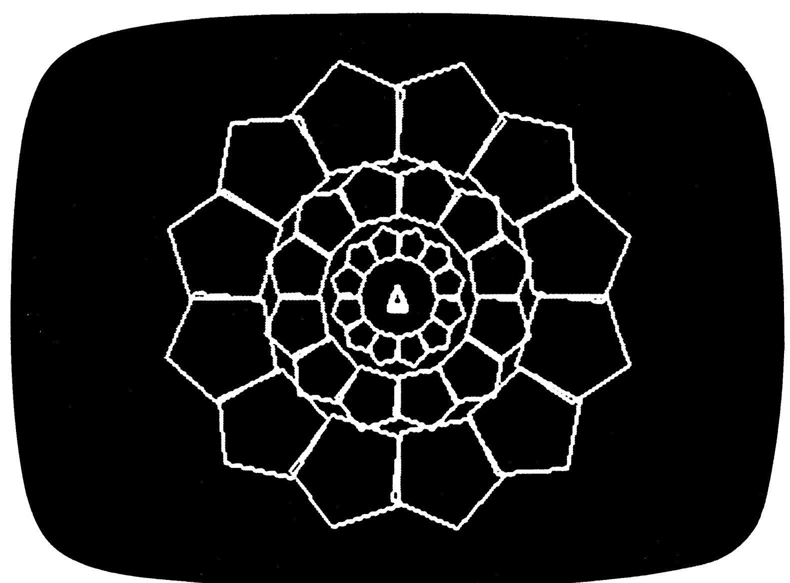
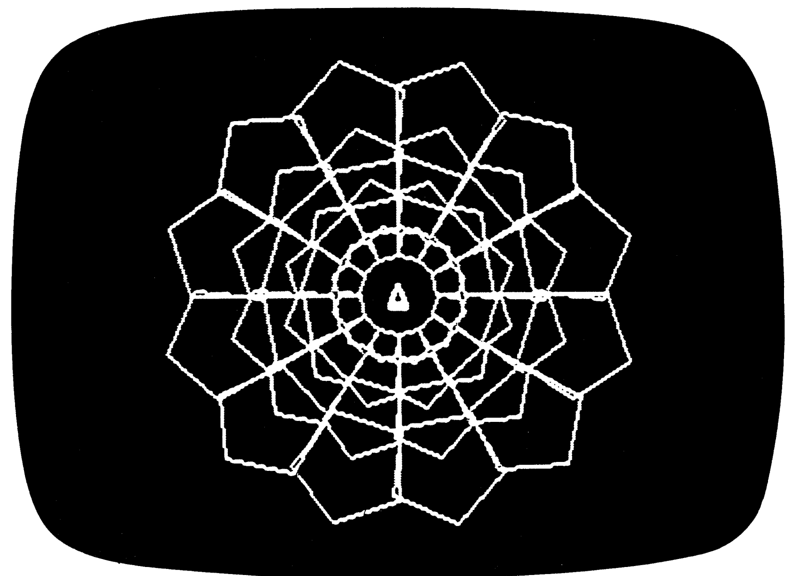
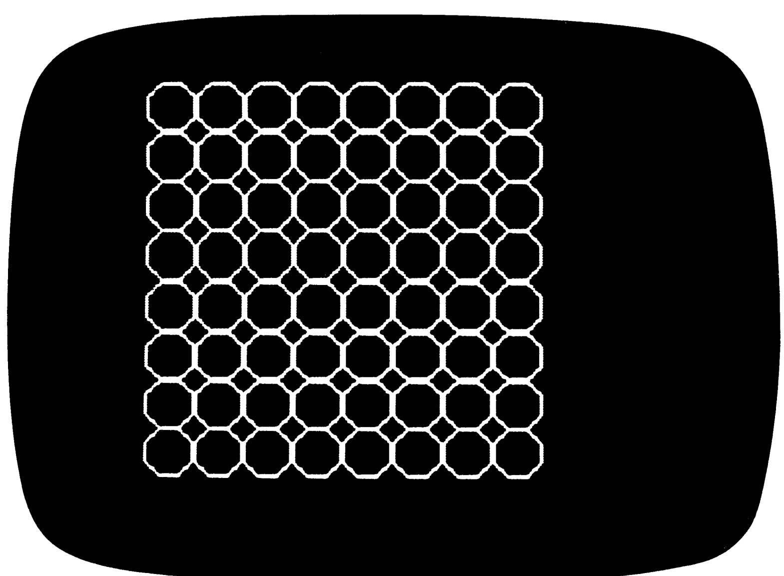


Figure 8-17. Another design drawn by PENT.12, with several values for :DISTANCE



You can also position the polygons in rows and columns, resulting in the type of pattern you might find on a tile floor. One such pattern, composed of octagons with sides 10 steps long, is illustrated in Figure 8-18. To get this pattern, use the procedure called OCT.TILE, which uses POLY and a new subprocedure called OCT.LINE. POLY, as we've seen, draws a polygon with a specified number of sides and a specified side length. OCT.LINE uses the REPEAT statement with POLY to draw a horizontal line of eight octagons (polygons with eight sides) 25 steps apart. OCT.TILE uses OCT.LINE eight times to draw eight lines of octagons, one underneath the other to create a tile pattern.

Figure 8-18. A tile design drawn by OCT.TILE



The OCT.TILE and OCT.LINE procedures look like this:

```
TO OCT.TILE
 FULLSCREEN BG 0
 PU SETXY (−125) 90 PD OCT.LINE
 PU SETXY (−125) 65 PD OCT.LINE
 PU SETXY (−125) 40 PD OCT.LINE
 PU SETXY (−125) 15 PD OCT.LINE
```

```
 PU SETXY (−125) (−10) PD OCT.LINE
 PU SETXY (−125) (−35) PD OCT.LINE
 PU SETXY (−125) (−60) PD OCT.LINE
 PU SETXY (−125) (−85) PD OCT.LINE
END

TO OCT.LINE
 FULLSCREEN BG 0
 REPEAT 8 [POLY 8 10 PU RT 90 FD 25 LT 90 PD]
END
```

You can get a different tile pattern by letting consecutive octagons overlap slightly. Just change the command FD 25 in the REPEAT statement of OCT.LINE to FD 18 and begin each new line of octagons 18 steps below the previous line instead of 25 steps below the previous line. Since this overlap makes each line of the design shorter, you can compensate by increasing the number of repetitions in OCT.LINE from 8 to 12 and the number of lines in OCT.TILE from 8 to 12. The new versions, called OCT.TILE.2 and OCT.LINE.2, produce the design shown in Figure 8-19 and are written like this:

```
TO OCT.TILE.2
 FULLSCREEN BG 0
 PU SETXY (−125) 90 PD OCT.LINE.2
 PU SETXY (−125) 72 PD OCT.LINE.2
 PU SETXY (−125) 54 PD OCT.LINE.2
 PU SETXY (−125) 36 PD OCT.LINE.2
 PU SETXY (−125) 18 PD OCT.LINE.2
 PU SETXY (−125) 0 PD OCT.LINE.2
 PU SETXY (−125) (−18) PD OCT.LINE.2
 PU SETXY (−125) (−36) PD OCT.LINE.2
 PU SETXY (−125) (−54) PD OCT.LINE.2
 PU SETXY (−125) (−72) PD OCT.LINE.2
 PU SETXY (−125) (−90) PD OCT.LINE.2
 PU SETXY (−125) (−108) PD OCT.LINE.2
END
```

```
TO OCT.LINE.2
 FULLSCREEN BG 0
 REPEAT 12 [POLY 8 10 PU RT 90 FD 18 LT 90 PD]
END
```

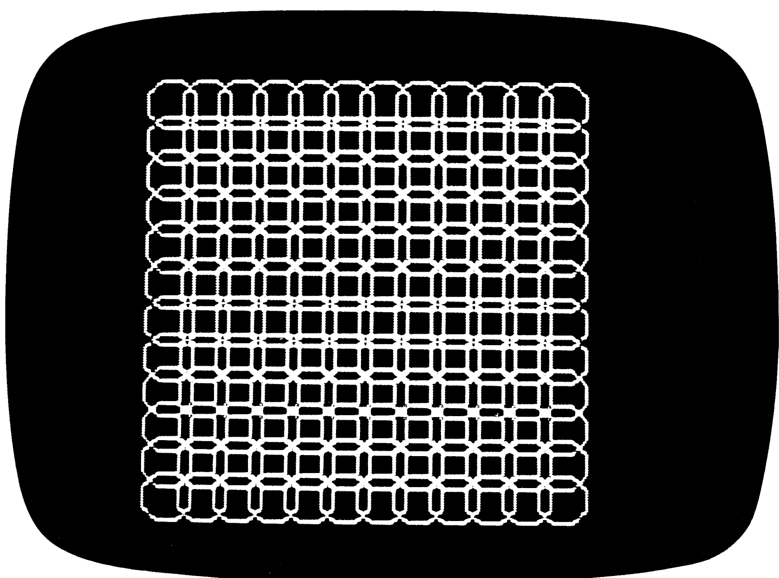


Figure 8-19. A tile design drawn by OCT.TILE.2

More Things to Do

Try varying the amount of overlap between successive polygons in the OCT.LINE procedure to see how the change affects the resulting design. You can also change the amount of overlap between successive lines of the design by changing the Y coordinates in the SETXY commands of OCT.TILE. You might also want to change the background and pen colors to see the effect on the resulting designs.

After you've experimented with the procedures OCT.TILE and OCT.LINE, see if you can develop similar procedures and tile designs using other kinds of polygons. Remember, you can save the procedures and the designs on your Work Disk. Use the SAVE command to save procedures and SAVEPICT to save drawings, as you did in Chapter 6.

Drawing Stars And Spirals

Using variables, as you did in Chapter 8, allowed you to draw striking wreath and tile patterns. In this chapter, you'll learn how to use variables to draw equally striking star and spiral patterns.

More important than simply teaching you how to draw stars and spirals, however, the activities in this chapter are based on two key mathematical operations—reducing fractions and using algebraic expressions.

Don't worry if your child hasn't yet encountered these mathematical operations at school. As with most Logo activities, your child doesn't need to understand the mathematical operations to write the procedures that draw stars and spirals. But, having used them in Logo, your child may find them easier to understand when they are introduced in school.

Drawing a Star

Although a polygon doesn't look much like a star, there are several similarities. A single modification in the POLY procedure used in Chapter 8 will result in a procedure that draws a star.

Unfortunately, modifying the POLY procedure won't always produce a star. The figure that is drawn will depend on the numbers you use as the variables. On the other hand, a fairly simple mathematical operation (reducing fractions) can be used to find the numbers that will produce stars.

In this section, you'll see the one modification you need to make in the POLY procedure to have it draw a star. Then, you'll learn about a few other ways to adjust the procedure that will make it one that's easy to use and that you'll want to save for use later. Finally, you'll learn how you can determine which numbers can be used in place of the procedure's variables in order to draw stars.

So, type in the POLY procedure if you don't already have it in your workspace or on your Work Disk. For convenience, here it is again:

```
TO POLY :NUMBER :SIDE
 FULLSCREEN BG 0
 REPEAT :NUMBER [FD :SIDE RT 360/:NUMBER]
END
```

Recall that :NUMBER is the variable for the number of sides of the polygon and :SIDE is the variable for the length of the sides. You can continue to use those same names for now,

though :NUMBER will represent the number of points on the star and :SIDE will represent the length of each line making the star. The polygon's corners will become the star's points.

To create a procedure that draws a star instead of a polygon, you want to multiply the angle of rotation within the REPEAT statement (360/:NUMBER) by 2. (I'll explain shortly why you multiply by 2.)

Here's what the new procedure, called STAR, looks like after that one modification of POLY:

```
TO STAR :NUMBER :SIDE
 FULLSCREEN BG 0
 REPEAT :NUMBER [FD :SIDE RT (360/:NUMBER)*2]
END
```

In Logo, multiplication is represented by the asterisk <*>, which is located above and to the right of the close bracket on your keyboard. To tell the computer you want to multiply the result of 360 divided by the number of lines, you need the parentheses around 360/:NUMBER, which represents the angle of each point.

Now, let's use both POLY and STAR so you can see the effect of the modification. Use 5 for the value of :NUMBER and 75 for the value of :SIDE. I suggest 5 because it produces a star when used in combination with 2, the number used for the multiplication factor.

Look at the figures produced by POLY 5 75 and STAR 5 75, first separately, and then on the screen at the same time. The angle of each turn in the original POLY was chosen specifically so that the total number of degrees of all the turns in the polygon would be 360 degrees, or exactly one revolution. Doubling the angle of each turn makes the total number of degrees 720, or two complete revolutions. You'll see that

doubling the angle of each turn results in a star with five points, instead of a pentagon with five sides, as in Figures 9-1, 9-2, and 9-3.

Figure 9-1. The shape drawn by POLY 5 75

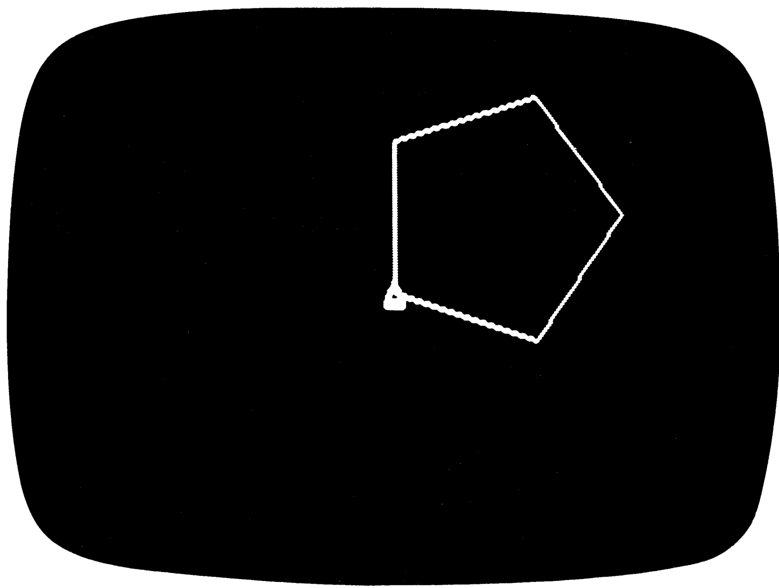


Figure 9-2. The shape drawn by STAR 5 75

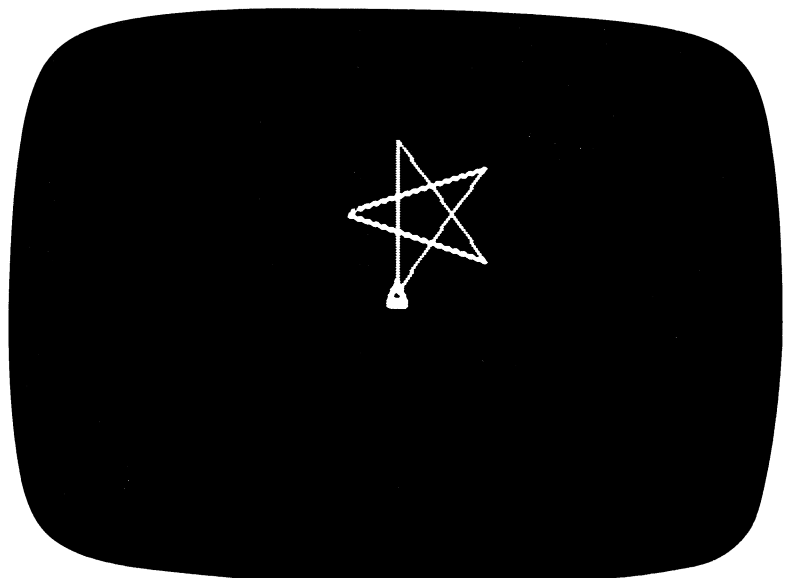
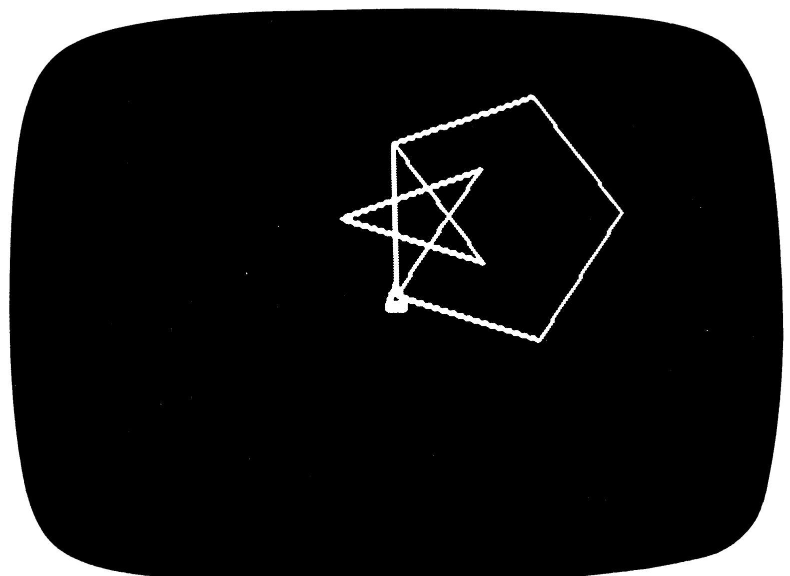
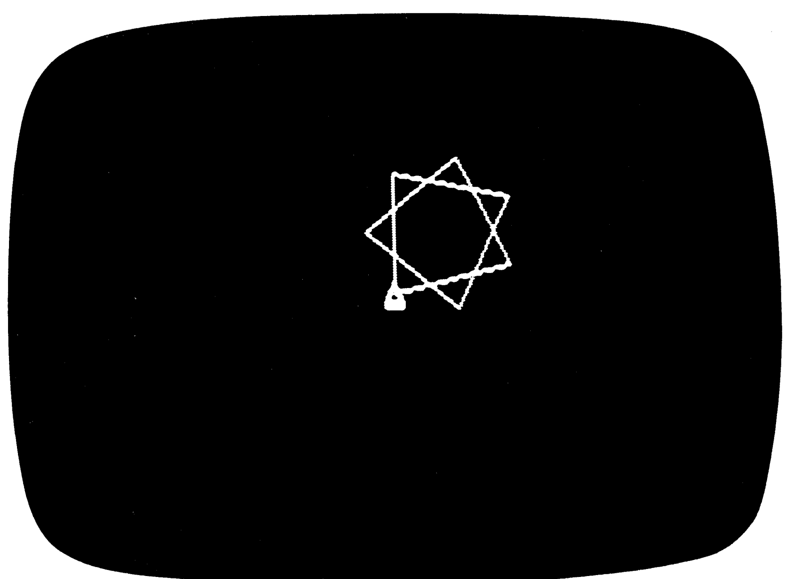


Figure 9-3. The shapes drawn by POLY 5 75 and STAR 5 75



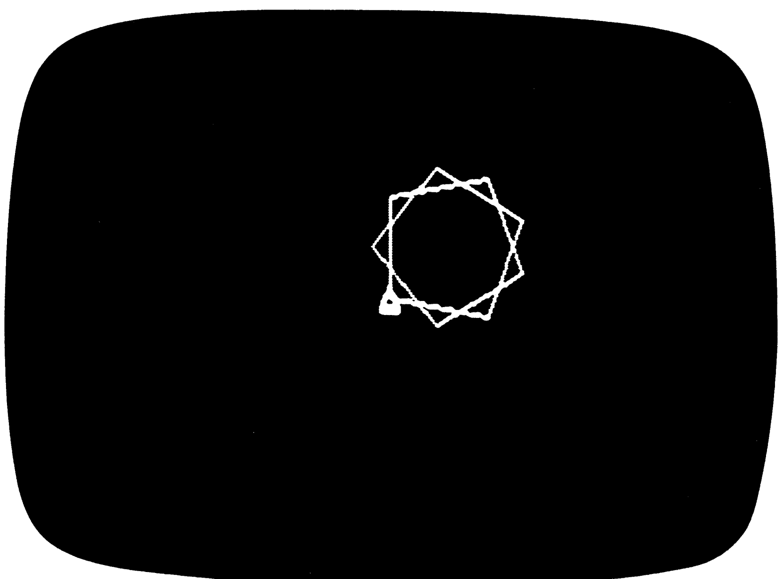
In fact, as you can verify for yourself with a few examples, multiplying by 2 results in a star when you choose an odd number 5 or greater for the value of :NUMBER. Figure 9-4 shows a star that was drawn by using 7 as the

Figure 9-4. The design drawn by STAR 7 60



value of :NUMBER and 60 as the value of :SIDE. Figure 9-5 shows a star that was drawn by using 9 as the value of :NUMBER and 50 as the value of :SIDE.

Figure 9-5. The design drawn by STAR 9 50



It would be handy if such a simple procedure as STAR always drew stars, but that's not the case. Only when :NUMBER is odd and is 5 or greater (5, 7, 9, 11, and so on) does STAR produce a star. Even numbers produce shapes other than stars. Figure 9-6 shows the drawing that results when :NUMBER is 10 and :SIDE is 60: a pentagon (a five-sided figure).

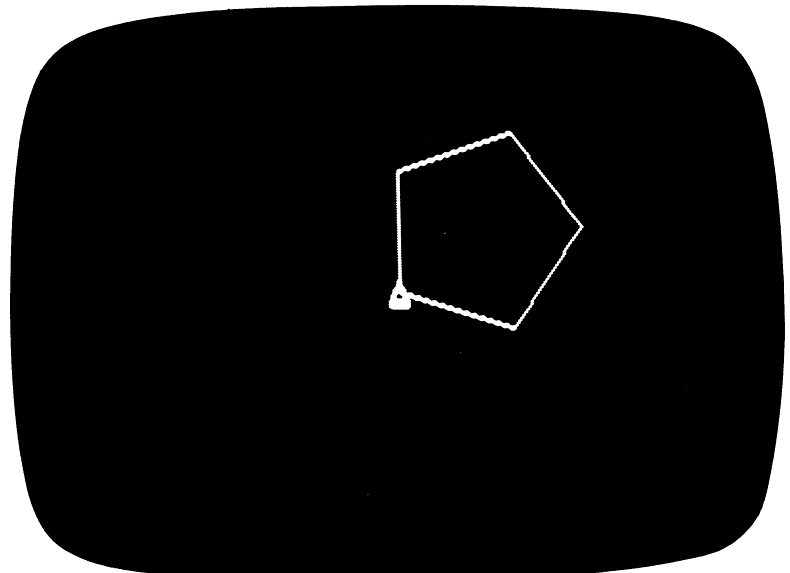


Figure 9-6. The shape drawn by STAR, with 10 as the value of :NUMBER

Why does STAR make such a distinction? To find out, you need to look at the other factor in deciding the shape of the resulting figure. You need to look at the number by which you multiply the angle of each point. The relationship between the number of sides, the number by which you multiply the angle of each point, and the resulting shape can be written mathematically as a fraction. To create the fraction, continue to use :NUMBER to represent the number of points on the star, but now use :MULTIPLIER instead of 2 to represent the amount by which the angle of each point is multiplied.

The following is a modified form of the STAR procedure, also called STAR, in which both :NUMBER and :MULTIPLIER are used as variables. For simplicity, the variable :SIDE in the previous version of STAR has been replaced by the constant value 60. Type this procedure so it will be available when you want to use it later in this chapter.

```
TO STAR :NUMBER :MULTIPLIER
 FULLSCREEN BG 0
 REPEAT :NUMBER [FD 60 RT (360/:NUMBER)
  *:MULTIPLIER]
END
```

To put the relationship between the variables in fraction form, make the number of points (:NUMBER) the numerator (the top line in a fraction) and the amount by which you multiply (:MULTIPLIER) the denominator (the bottom line in a fraction). For instance, in the example used earlier that led to the pentagon shown in Figure 9-6, the value of :NUMBER is 10 and the value of :MULTIPLIER is 2, so the resulting fraction is 10/2.

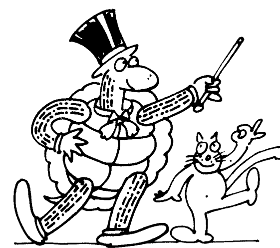
Reduce the fraction :NUMBER/:MULTIPLIER to its lowest common form. Lowest common form means 1 is the only number you can divide both the numerator and denominator by and obtain a whole number (as opposed to a fraction). In our example, the fraction 10/2 can be reduced to 5/1 as its lowest common form.

In this reduced form, if the denominator (:MULTIPLIER) is 1 or :NUMBER−1, then the resulting figure will not be a star. In other words, :NUMBER should always be at least 2 larger than :MULTIPLIER in order for the procedure to work. In our example, the reduced form 5/1 does have a denominator of 1. That's why a pentagon was drawn instead of a star. But if the denominator (:MULTIPLIER) is not 1 or :NUMBER−1, then the resulting figure will be a star.

If :NUMBER and :MULTIPLIER have no common divisors, and :MULTIPLIER is not 1 or :NUMBER−1, then the fraction already is in its reduced form and the resulting figure will be a star. That's why STAR draws a star only when the number of sides is odd and at least 5; 2 has no common divisors with any odd number and it obviously isn't equal to either 1 or 1 less than any number larger than 5.

If :NUMBER is even, you still can use STAR to draw a star if you can find a number for :MULTIPLIER that has no common divisors with the number you're using for :NUMBER. For example, if :NUMBER is 8, then multiplying by 3 (:MULTIPLIER = 3) will produce an eight-pointed star, since 8 and 3 have no common divisors and 3 is not equal to 8−1.

If :NUMBER is 12, then 3 would not work as :MULTIPLIER, since 12 and 3 have 3 as a common divisor.



Turtle Tip: The relationship in STAR between the two variables and the resulting figure not only provides a good opportunity for your child to practice reducing fractions, but also shows what it means for two fractions to be equivalent. Two fractions are equivalent if they have the same reduced form.

Suppose, for example, your child wants to use STAR to draw a 10-pointed star. According to the rule on turning polygons into stars, if :NUMBER is 10, the number your child picks for :MULTIPLIER must render an already-reduced form of the fraction 10/:MULTIPLIER. If your child selects an incorrect value for :MULTIPLIER, the mistake should be apparent as soon as the resulting drawing appears on the screen. You can then help with finding out where the error occurred and why STAR didn't produce a 10-pointed star. For example, if your child selected 4 as :MULTIPLIER and obtained a five-pointed star, you could point out that 10/4 can be reduced to 5/2 and the five-pointed star is exactly what is predicted by the rule governing whether a particular pair of numbers produces a star.

You can also use STAR to play a game of prediction with your child. Take turns giving each other a pair of numbers, one number for :NUMBER and one for :MULTIPLIER, trying to predict whether STAR will draw a star. First, form a fraction from the two numbers, reducing it to its lowest form if it isn't in its lowest form already. Then, apply the rule on drawing stars to predict what the figure will be. Test the prediction by using STAR with the numbers you picked.

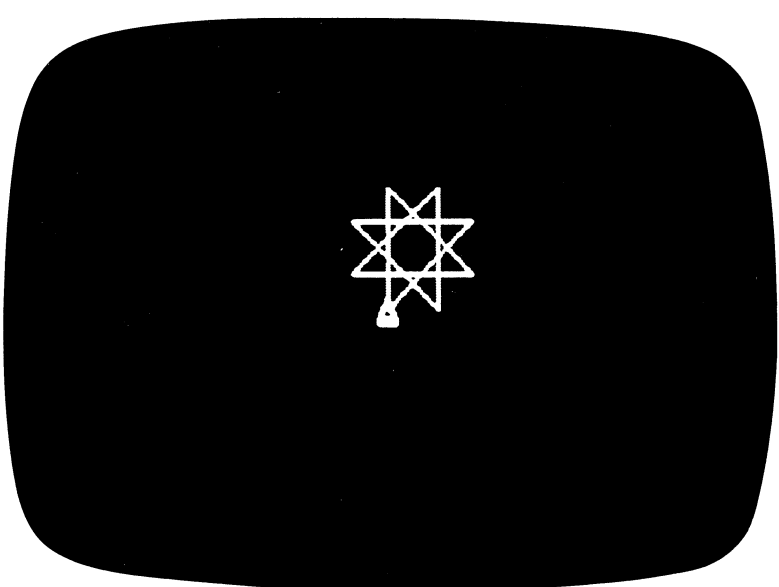
For a variation on the prediction game, pick two pairs of numbers, two numbers for :NUMBER and two for :MULTIPLIER. Put each pair in fraction form, and then ask your child to try to tell whether the two fractions are equivalent—that is, if both have the same reduced form. Your child can test the prediction by using the two pairs of numbers in STAR to see if both pairs of numbers produce the same figure. If the two pairs of numbers produce the same figure, their fractions have the same reduced form—in other words, they are equivalent.

The Logo Activity Time activity at the end of this chapter, called FRACTIONS, is a game that uses equivalent fractions. Try it when you come to the end of the chapter and see how Logo can be used to make learning about and practicing mathematics enjoyable.

But 5 would work as :MULTIPLIER, since 12 and 5 have no common divisors and 5 is not equal to 12−1.

Use STAR now, with appropriate values for :NUMBER and :MULTIPLIER, to draw stars having 8, 10, 12, and 14 points, as illustrated in Figures 9-7, 9-8, 9-9, and 9-10. If you want to, you can edit STAR so that the length of the lines in the star is also a variable, as in the earlier version of STAR. That way, you can draw stars in different sizes as well as stars with different numbers of points.

Figure 9-7. An 8-pointed star drawn by STAR



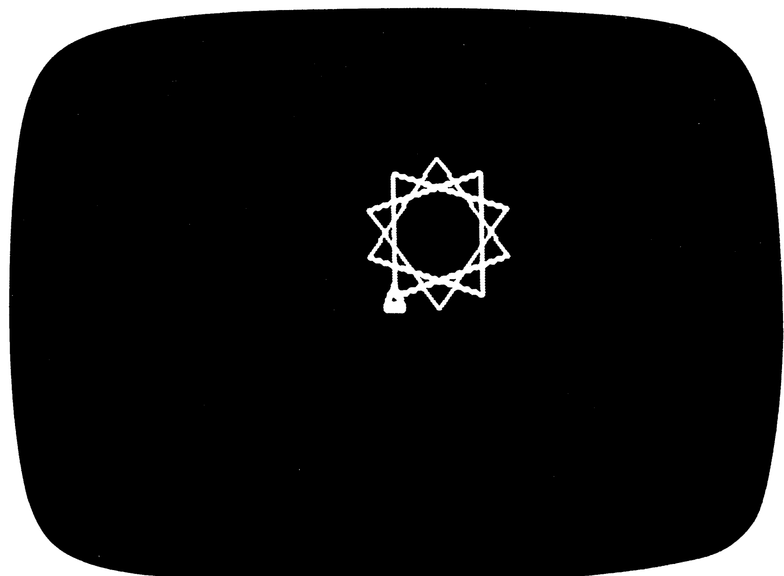


Figure 9-8. A 10-pointed star drawn by STAR

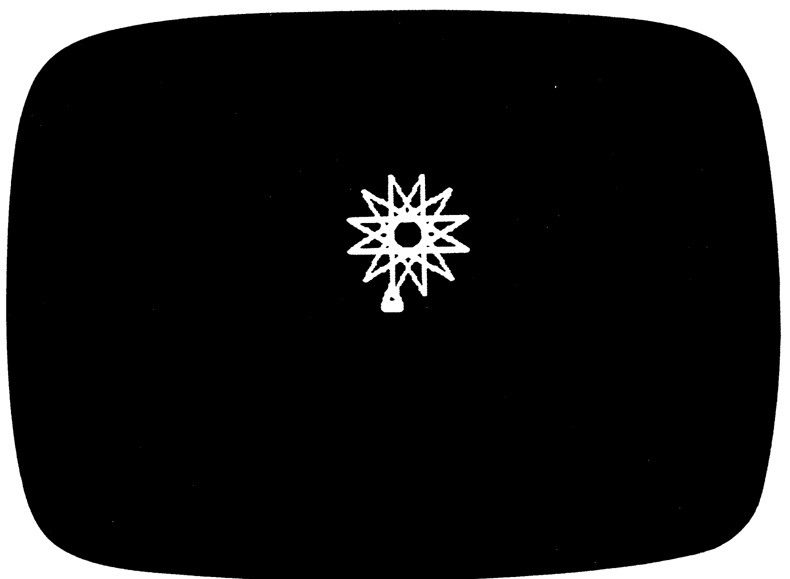
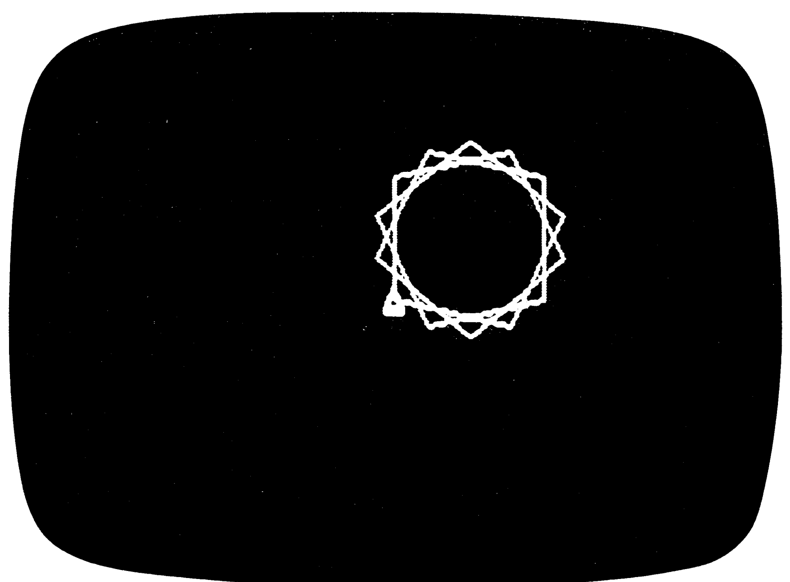


Figure 9-9. A 12-pointed star drawn by STAR

Figure 9-10. A 14-pointed star drawn by STAR



More Things to Do

Now that you've had some practice using STAR to draw stars with different numbers of points, you should be able to develop wreath and tile patterns composed of stars the same way you developed wreath and tile patterns of polygons in Chapter 8. Go ahead and try. Remember, you can take advantage of the various background and pen colors, as well as the size and number of points of the stars, in developing designs. Just to give you an idea of the kind of design that is possible using stars, the design illustrated in Figure 9-11, produced by the procedure WREATHS.4, is composed of four star-wreaths, with alternating green and red wreaths. WREATHS.4 uses the subprocedures STAR.WREATH and STAR.2. STAR.2 is the same as STAR except that STAR.2 uses the variable :SIDE for the lengths of the lines in the stars it draws.

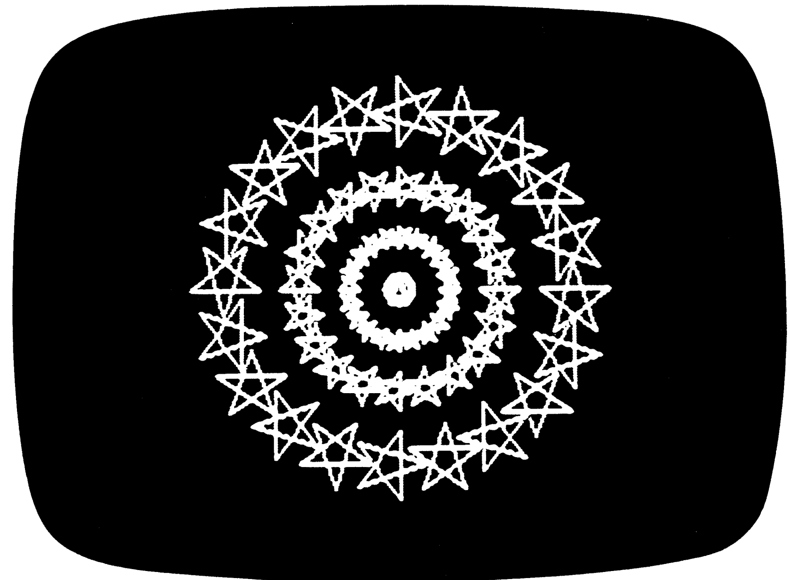


Figure 9-11. A star-wreath design drawn by WREATHS.4

```
TO WREATHS.4
 FULLSCREEN BG 0
 HT
 PC 5
 STAR.WREATH 70
 STAR.WREATH 20
 PC 2
 STAR.WREATH 40
 STAR.WREATH 5
 ST
END

TO STAR.WREATH :DISTANCE
 FULLSCREEN BG 0
 REPEAT 20 [PU FD :DISTANCE PD STAR.2 5 2
  :DISTANCE/2 PU BK :DISTANCE RT 18 PD]
END

TO STAR.2 :NUMBER :MULTIPLIER :SIDE
 FULLSCREEN BG 0
 REPEAT :NUMBER [FD :SIDE RT (360/:NUMBER)
  *:MULTIPLIER]
END
```

Using the Make Statement

Listing variable names on the same line as the procedure name is one way to define variable names in Logo. But there's another way—with the MAKE statement.

The MAKE statement allows you to write procedures containing variables that change value as the procedure runs. That, in turn, gives you much greater power and flexibility in the procedures you can write and in the results these procedures can produce.

To use the MAKE statement, type the word MAKE, a space, the variable name you want to use, another space, then the value you want the variable to have. The only requirement in using the MAKE statement is that the variable name must begin with a quotation mark. In Logo, a quotation mark before a letter or a word tells the computer you're referring to the name of the variable. A colon before a letter or a word tells the computer you're referring to the value of a variable.

The statement MAKE "X 4 has the double effect of defining a new variable called "X and giving this variable a value of 4.

You can also use the MAKE statement to change the value of a variable that's already been given a value. For example, you could type MAKE "X 4 to give "X a value of 4, and then type MAKE "X 5 to change the value of "X to 5. You could also change the value of "X from 4 to 5 by typing MAKE "X 4 and then MAKE "X :X+1.

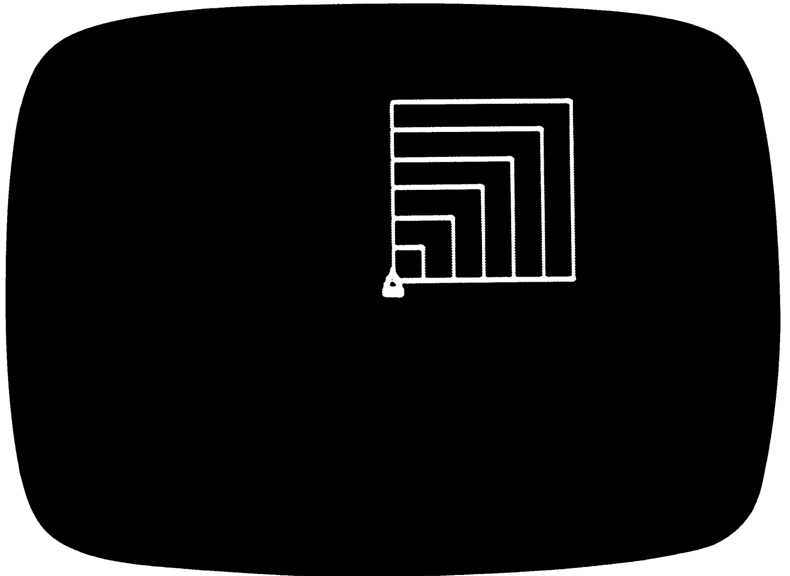
The statement MAKE "X :X+1 tells the computer to take the variable name "X and make it one more than its current numerical value. The numerical value of "X (which is :X) is 4 in the first statement, so making "X one larger (:X+1) changes its value from 4 to 4+1, or 5.

This procedure, SQUARES, which uses the SQUARE subprocedure, demonstrates the use of a MAKE statement:

```
TO SQUARES :NUMBER
 FULLSCREEN BG 0
 MAKE "SIDE 15
 REPEAT :NUMBER [SQUARE :SIDE MAKE "SIDE :SIDE+15]
END

TO SQUARE :SIDE
 FULLSCREEN BG 0
 REPEAT 4 [FD :SIDE RT 90]
END
```

When using SQUARES, you specify the number of squares you want drawn and SQUARES draws them, with the first square having sides 15 steps long and each succeeding square having sides 15 steps longer each time. The key to being able to make each square larger is the inclusion of MAKE "SIDE :SIDE+15 at the end of the REPEAT statement. The MAKE statement causes the value of "SIDE to increase from 15 to 30, to 45, and so on. For example, SQUARES 6 gives the six squares illustrated in Figure 9-12.

Figure 9-12. The design drawn by SQUARES 6



You can draw a more symmetrical picture if you begin each square at screen coordinates X= –:SIDE/2 and Y= –:SIDE/2. (That is, the starting X coordinate is left of the center of the screen, half the length of the side of the square; and the starting Y coordinate is below the center of the screen, half the length of the side of the square.) The squares produced are centered around the Home position. The following modification of SQUARES, called SQUARES.2, does that:

```
TO SQUARES.2 :NUMBER
 FULLSCREEN BG 0
 MAKE "SIDE 15
 REPEAT :NUMBER [PU SETXY (–:SIDE/2) (–:SIDE/2)
  PD SQUARE :SIDE MAKE "SIDE :SIDE+15]
END
```

Typing SQUARES.2 15 results in 15 symmetrical squares, as illustrated in Figure 9-13.

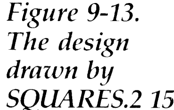
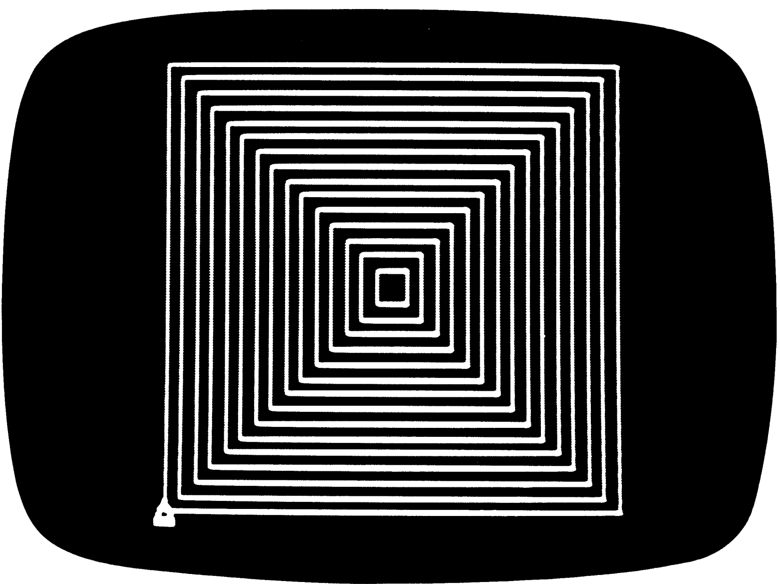


Figure 9-13. The design drawn by SQUARES.2 15



Of course, you can also develop procedures like SQUARES and SQUARES.2 using polygons other than squares. For example, the following procedures, all of which include the word HEXS in their names, use hexagons instead of squares. HEXS draws increasingly larger hexagons, all starting at the Home position, as in Figure 9-14, which illustrates HEXS 5. HEXS.2 draws increasingly larger hexagons which are all symmetrically centered around the Home position, as in Figure 9-15, which illustrates HEXS.2 8. Both procedures make use of the subprocedure POLY developed earlier. For ease of reference, the POLY procedure is listed here as well.

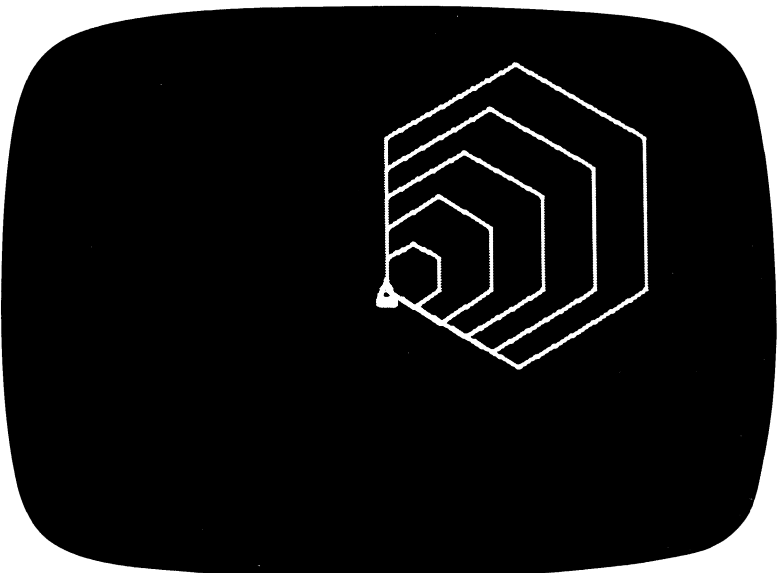
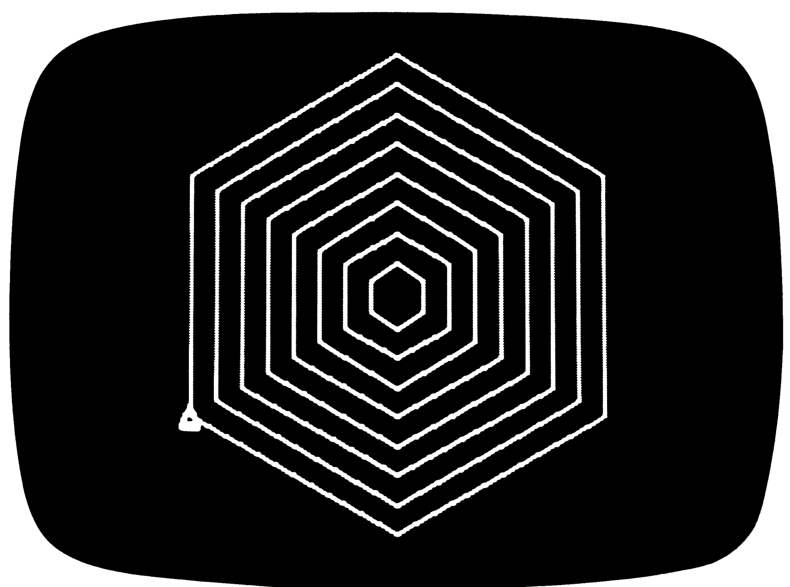


Figure 9-14. The design drawn by HEXS 5

Figure 9-15. The design drawn by HEXS.2 8



```
TO HEXS :NUMBER
 FULLSCREEN BG 0
 MAKE "SIDE 15
 REPEAT :NUMBER [POLY 6 :SIDE MAKE "SIDE :SIDE+15]
END

TO HEXS.2 :NUMBER
 FULLSCREEN BG 0
 MAKE "SIDE 15
 REPEAT :NUMBER [PU LT 120 FD 15 RT 120 PD
  POLY 6 :SIDE MAKE "SIDE :SIDE+15]
END

TO POLY :NUMBER :SIDE
 FULLSCREEN BG 0
 REPEAT :NUMBER [FD :SIDE RT 360/:NUMBER]
END
```

The MAKE statement can be used in a variety of ways to simplify the writing of complicated procedures. For example, OCT.TILE, a procedure developed earlier, draws a tile pattern composed of octagons. The OCT.TILE procedure was rather long because, every time you wanted to set the turtle in motion drawing a new line of octagons below the previous line of octagons, you had to use a SETXY command to position it correctly. Using a MAKE statement, you can condense the OCT.TILE procedure considerably. The following is a condensed version of OCT.TILE, together with the OCT.LINE subprocedure it uses. (OCT.LINE uses POLY as a subprocedure, so make sure you've defined POLY and it's currently in your workspace by asking the computer to display a list of the titles of all the procedures in the workspace, using the command PRINTOUT TITLES (or POTS), as you did in Chapter 6.)

```
TO OCT.TILE
 FULLSCREEN BG 0
 MAKE "Y 90
 REPEAT 8 [PU SETXY (-125) (:Y) PD OCT.LINE MAKE
  "Y :Y-25]
END

TO OCT.LINE
 FULLSCREEN BG 0
 REPEAT 8 [POLY 8 10 PU RT 90 FD 25 LT 90 PD]
END
```

Viewing Your Variable Names

If you want to know which variable names you have in your workspace at any particular time, and the values that are currently assigned to them, you can use the command PRINTOUT NAMES (or PO NAMES) in the NoDraw mode. A list of all the variable names and values will be printed out on your display screen. For example, suppose you're adding new procedures to an existing file and, for consistency, you

want to use the same variable names in the new procedures that you used in the earlier ones. Or, suppose a certain variable changes its value during the running of a particular procedure and you want to know the value of that variable after the procedure has ended. In either case, typing PRINTOUT NAMES (or PO NAMES) will give you the desired information. Variable names and their current values are automatically saved when you save your workspace as a file on a disk, but since variables take up space on the disk, you may want to erase some or all of the variable names and values before saving the procedures. To erase variable names, use the command ERASE NAMES (or ER NAMES).

More Things to Do

After you wrote the OCT.TILE procedure, which produced a tile pattern of octagons that didn't overlap, you wrote a modified version called OCT.TILE.2, which produced a tile pattern with overlapping octagons. Use the MAKE statement to write a condensed version of OCT.TILE.2 and then run the new version to make sure it works.

Another nice use of the MAKE statement is changing the turtle's pen color while drawing a design, giving different parts of the design different colors. The following modified version of the SQUARES.2 procedure, called SQUARES.3, uses MAKE in just that way to draw concentric squares of different colors.

```
TO SQUARES.3 :NUMBER
 FULLSCREEN BG 0
 MAKE "SIDE 15
 MAKE "COLOR 1
 REPEAT :NUMBER [PU SETXY (-:SIDE/2) (-:SIDE/2)
  PD PC :COLOR SQUARE :SIDE MAKE "SIDE
  :SIDE+15 MAKE "COLOR :COLOR+1]
END
```

Turtle Tip: The MAKE statement is an important and powerful tool in the development of Logo graphics, as you've already seen, and non-graphics, as you'll see in later chapters. But young children sometimes have difficulty grasping the MAKE statement because of its condensed form. This condensed form is understandable to the computer and saves time in typing, but it tends to hide what the statement really does. So, you may need to help your children understand what the statement does and how to interpret it when it appears within a procedure.

One way of helping your children understand what the MAKE statement does is to say out loud, in an expanded form, what the statement does. For example, if you're typing the command MAKE "X 4, say out loud as you type, "Make the name "X represent the number 4." And if you see the statement MAKE "SIDE :SIDE + 3, say out loud, "Make the name "SIDE increase from its present value :SIDE to 3 greater." Saying aloud what the MAKE statement represents will make the meaning more clear to you and your children, helping you learn to use the MAKE statement correctly.

The MAKE statement in Logo is similar to the variable in algebra, a mathematical expression that children often have trouble with in secondary school. Standard algebraic expressions like "Let X represent the distance traveled by a car in 4 hours," or "Let X be increased by 4 to X + 4" use variable names the same way the MAKE statement does. Because of this similarity between Logo and algebra, your children can use Logo's MAKE statement to gain an understanding of variable names in algebra before actually encountering them in the classroom.

In fact, the wording I've suggested you use when you say what a MAKE statement means is taken directly from the way variables are described in algebra. Using the same wording as is used in algebra will help your children develop a familiarity with both the use of algebraic variables and the terminology employed in the classroom.

SQUARES.3 uses the variable name "COLOR to represent the turtle's pen color and starts drawing in white (color number 1) with the statement MAKE "COLOR 1. The statement MAKE "COLOR :COLOR + 1 changes the pen color to red (color number 1 + 1, or 2). Each new square is drawn in the color with the next higher number, so that the third square is drawn in cyan (color number 3), the fourth square is drawn in purple (color number 4), and so on. The command

PC :COLOR tells the computer the variable "COLOR refers to the pen color.

Try SQUARES.3 with different numbers of squares to see how much more colorful the final drawing becomes with the addition of these few extra statements.

After you've run SQUARES.3 a few times, see if you can use the MAKE statement to modify HEXS.2 and OCT.TILE the same way. In OCT.TILE, for example, make each line of octagons a different color. Keep in mind, though, that you don't have to limit yourself to always starting with white (color number 1) and always increasing the pen color number by 1 each time. You can just as easily begin with colors other than white and increase the pen color number by 2, or 3, or 4 each time. Or, you can start with a large color number, such as 10 (light red), and decrease the number by 1, 2, or 3 each time.

Drawing Spirals with the Make Statement

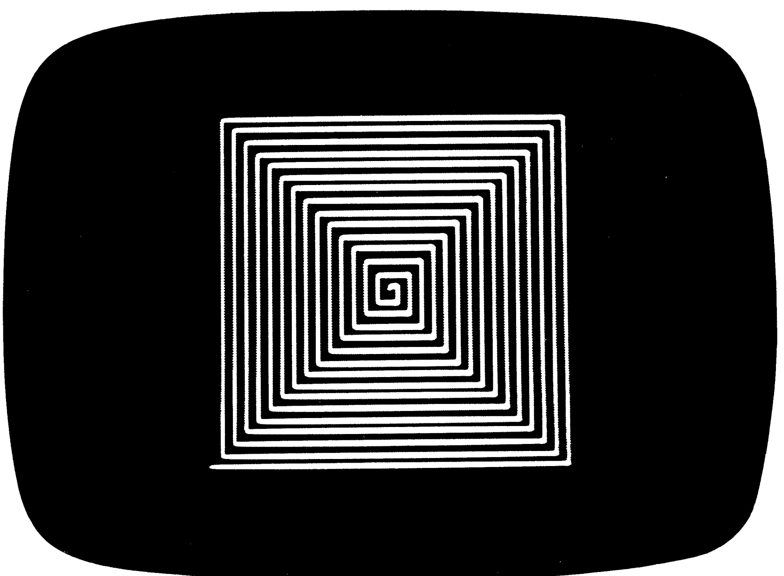
A spiral is like a circle in that it revolves around a central position. However, while any point on a circle is the same distance from the circle's center as any other point, points on a spiral move farther and farther from the center as the spiral goes around and around.

In Logo, it's a simple matter to use the MAKE statement to draw a spiral. You just start the turtle at the center of the screen and use the MAKE statement to increase the turtle's distance from the center as it moves around. For example, the following procedure, called SQ.SPIRAL, draws a spiral of lines and corners.

```
TO SQ.SPIRAL
 FULLSCREEN BG 0 HT
 MAKE "SIDE 1
 REPEAT 60 [FD :SIDE RT 90 MAKE "SIDE :SIDE+3]
END
```

SQ.SPIRAL starts out as one side of a square, but each successive side is three steps longer than the previous side. The variable "SIDE is defined inside the procedure using the MAKE statement and given the initial value of 1. The name of the variable is "SIDE, and the value of the variable is :SIDE, which increases by 3 each time another side of the figure is drawn. It's this increase in the length of the sides that causes the turtle to move away from the center of the screen and results in a spiral with a total of 60 sides. The procedure includes a command to hide the turtle so you can more easily see the spiral as it's being drawn. The completed spiral is shown in Figure 9-16.

Figure 9-16. The spiral drawn by SQ.SPIRAL



Even more beautiful, are the spiral designs you can obtain if, instead of turning the turtle 90 degrees before drawing each new side of the figure, you turn it some other number of degrees. The following modified version of SQ.SPIRAL,

called SQ.SPIRAL.2, does that by using "ANGLE as the name of the variable:

```
TO SQ.SPIRAL.2 :ANGLE
 FULLSCREEN BG 0 HT
 MAKE "SIDE 1
 REPEAT 60 [FD :SIDE RT :ANGLE MAKE "SIDE :SIDE+3]
END
```

Figures 9-17 through 9-20 illustrate the effect on the resulting spiral design of varying the angle of each corner from 87 degrees to 89 degrees to 91 degrees to 93 degrees.

Figure 9-17. The spiral drawn by SQ.SPIRAL.2, with each corner 87 degrees

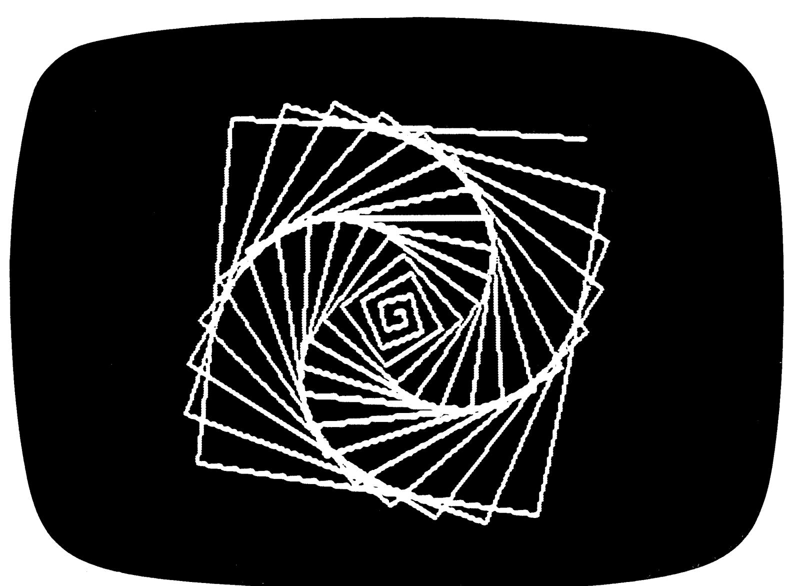


Figure 9-18. The spiral drawn by SQ.SPIRAL.2, with each corner 89 degrees

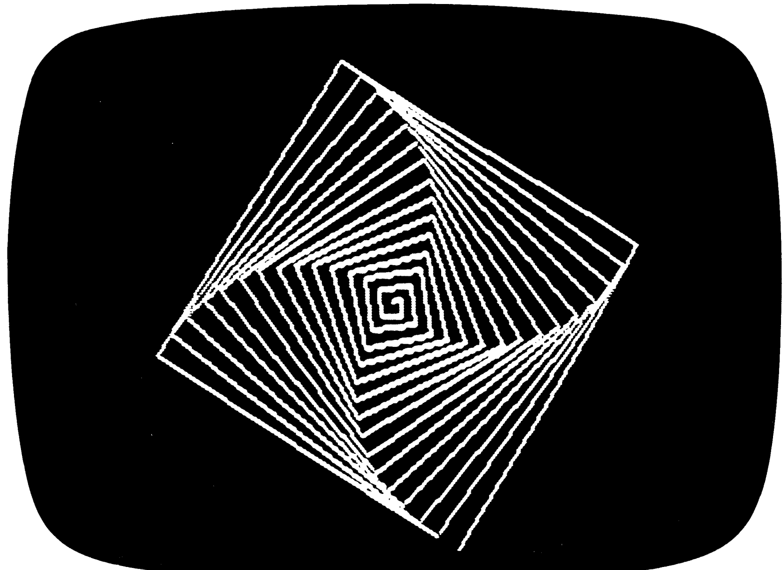
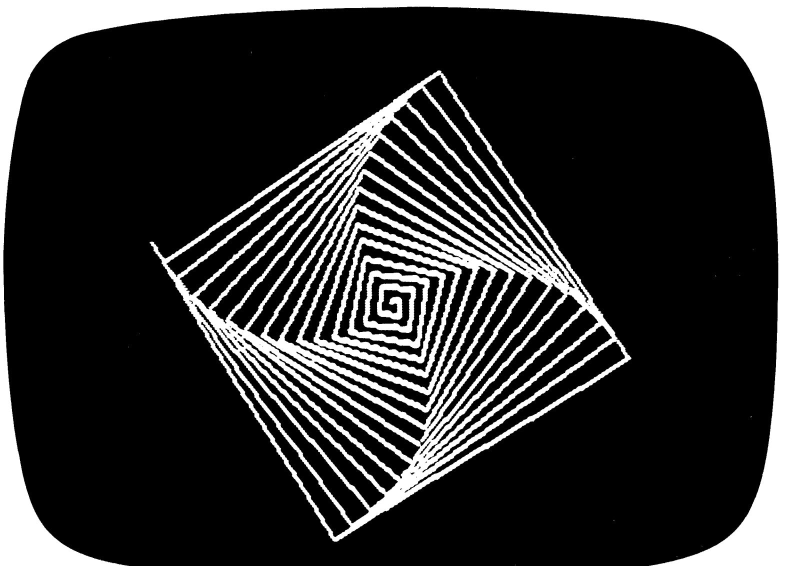


Figure 9-19. The spiral drawn by SQ.SPIRAL.2, with each corner 91 degrees



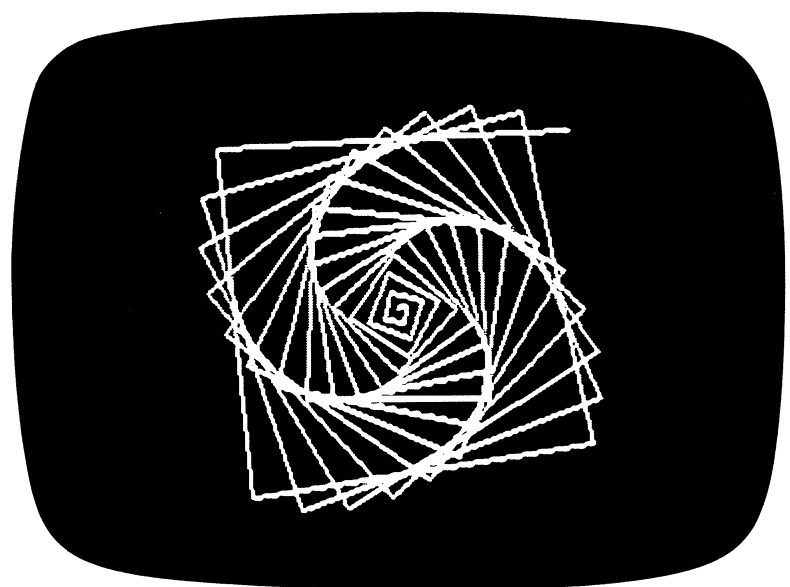


Figure 9-20. The spiral drawn by SQ.SPIRAL.2, with each corner 93 degrees

You can also use the MAKE statement to draw successive parts of the spirals in different pen colors. For example, suppose you want to color each of the designs in a different one of the six rainbow colors: red, orange, yellow, green, blue, and purple, in that order. The following procedure, called SQ.RAINBOW, and its subprocedure, SQ.SPIRAL.3, do that.

```
TO SQ.RAINBOW :ANGLE
 FULLSCREEN BG 0 HT
 MAKE "SIDE 1
 SQ.SPIRAL.3 :ANGLE 2
 SQ.SPIRAL.3 :ANGLE 8
 SQ.SPIRAL.3 :ANGLE 7
 SQ.SPIRAL.3 :ANGLE 5
 SQ.SPIRAL.3 :ANGLE 6
 SQ.SPIRAL.3 :ANGLE 4
END

TO SQ.SPIRAL.3 :ANGLE :COLOR
 FULLSCREEN BG 0
 PC :COLOR
 REPEAT 10 [FD :SIDE RT :ANGLE MAKE "SIDE :SIDE+3]
END
```

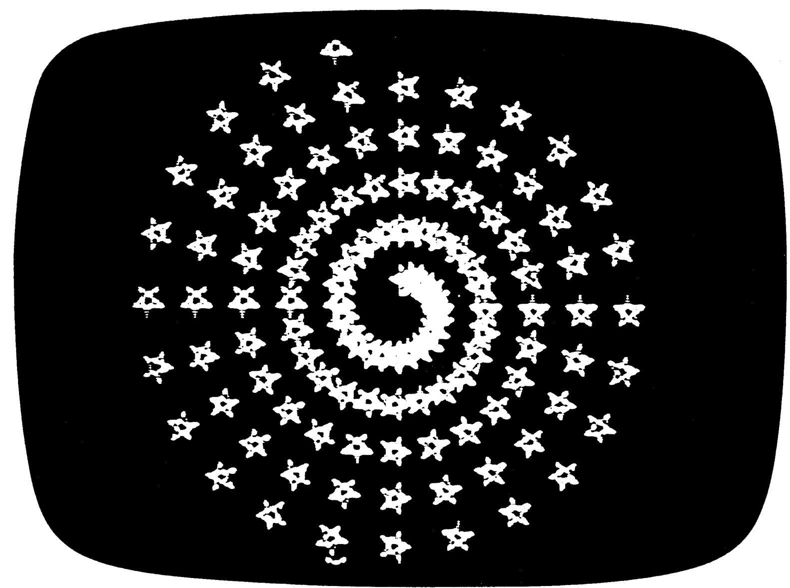
The SQ.SPIRAL.3 subprocedure uses variable names for both the angle of each corner and the pen color, but the number of sides is a constant 10, which is one-sixth of the total figure. This allows the major procedure, SQ.RAINBOW, to give the variable "SIDE the beginning value 1, and then call on SQ.SPIRAL six times, giving it a new pen color each time to draw the next 10 sides of the spiral. Try SQ.RAINBOW with the same angles for each corner as you used before (87, 89, 91, and 93 degrees) to see the effect of incorporating pen color changes within a procedure using the MAKE statement.

Expanding Your Spiral Designs

Another way of drawing a spiral is to have the turtle draw a sequence of geometric figures, with each successive figure positioned further and further away from the center of the screen. The following procedure, STAR.SPIRAL, uses this approach to draw a spiral composed of 120 stars. Each star has five points and each of its lines is 15 steps long. All are drawn in purple on a black background by the STAR.5 subprocedure. The first star is drawn five steps from the center of the screen. To draw each successive star, the turtle turns 15 degrees to the right and moves one step away from the center. The star spiral produced by this procedure is shown in Figure 9-21.

```
TO STAR.SPIRAL
 FULLSCREEN BG 0 PC 4 HT
 MAKE "DISTANCE 5
 REPEAT 120 [PU FD :DISTANCE PD STAR.5 PU BK
  :DISTANCE RT 15 MAKE "DISTANCE :DISTANCE+1]
END

TO STAR.5
 FULLSCREEN BG 0
 REPEAT 5 [FD 15 RT (360/5)*2]
END
```

Figure 9-21. The star spiral drawn by STAR.SPIRAL



As you did with SQ.RAINBOW, you can modify STAR.SPIRAL so that successive parts of the spiral are drawn in the different colors. One way to do this is to modify STAR.SPIRAL so that it draws only 20 stars, then changes the turtle's pen color before drawing a new set of 20 stars. The following procedure, called STAR.RAINBOW, does that. STAR.RAINBOW uses a modified version of STAR.SPIRAL, called STAR.SPIRAL.2, as a subprocedure. STAR.5 is required as a subprocedure, so make sure it is still in your workspace.

```
TO STAR.RAINBOW
 FULLSCREEN BG 0 HT
 MAKE "DISTANCE 5
 PC 2 STAR.SPIRAL.2
 PC 8 STAR.SPIRAL.2
 PC 7 STAR.SPIRAL.2
 PC 5 STAR.SPIRAL.2
 PC 6 STAR.SPIRAL.2
 PC 4 STAR.SPIRAL.2
END
```

```
TO STAR.SPIRAL.2
 FULLSCREEN BG 0
 REPEAT 20 [PU FD :DISTANCE PD STAR.5 PU BK
   :DISTANCE RT 15 MAKE "DISTANCE :DISTANCE+1]
END
```

More Things to Do

Edit the SQ.SPIRAL and SQ.SPIRAL.2 procedures so that, instead of increasing the "SIDE variable by 3 each time, they increase "SIDE by a variable amount with the "INCREMENT variable name. Then, run these procedures with different positive numbers for "INCREMENT to see the effect of changing the value on the spiral. You'll find that the larger the number you use in place of "INCREMENT, the more spread-out the spiral is, and the smaller the number you use in place of "INCREMENT, the more compact the spiral is.

You might also try developing procedures similar to SQ.SPIRAL, SQ.SPIRAL.2, and SQ.RAINBOW that use triangles instead of squares. Call these procedures TRI.SPIRAL, TRI.SPIRAL.2, and TRI.RAINBOW; and use an angle of rotation of 120 degrees for the basic TRI.SPIRAL procedure. For a spiral that fits nicely on your screen and is neither too compact nor too spread out, try using 35 repetitions and an increment of six steps for each successive side of the spiral, instead of the 60 repetitions and increment of three steps you used with the square-shaped spirals. Using TRI.SPIRAL.2 with angles of rotation of 117, 119, 120, 121, and 123 degrees, you'll obtain the spiral designs illustrated in Figures 9-22 through 9-26.

Figure 9-22. The spiral drawn by TRI.SPIRAL.2, with each corner 117 degrees

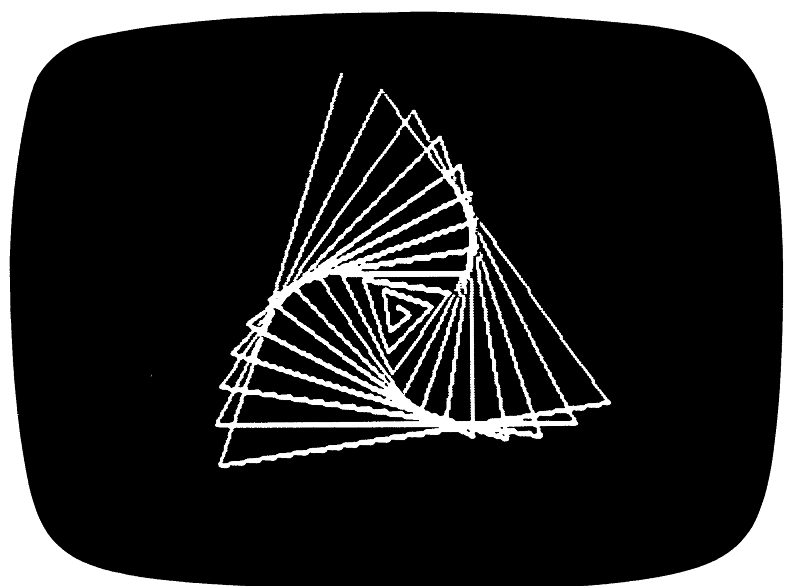
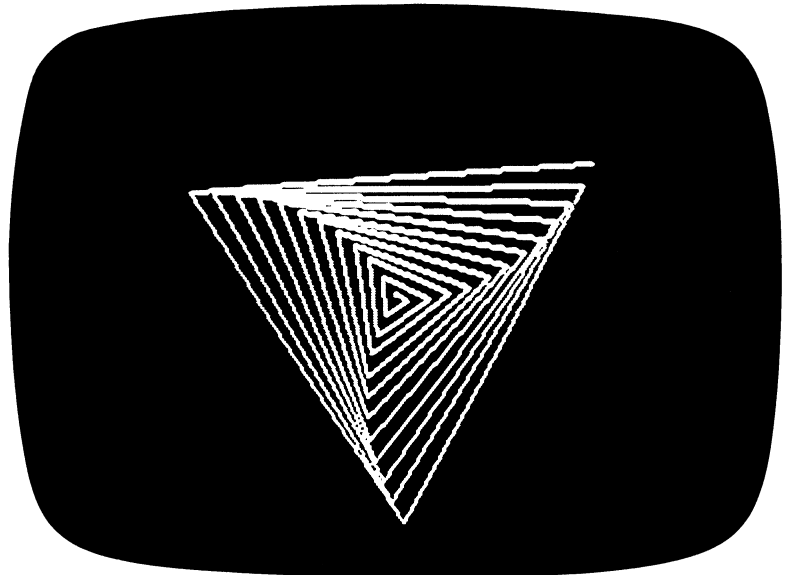


Figure 9-23. The spiral drawn by TRI.SPIRAL.2, with each corner 119 degrees



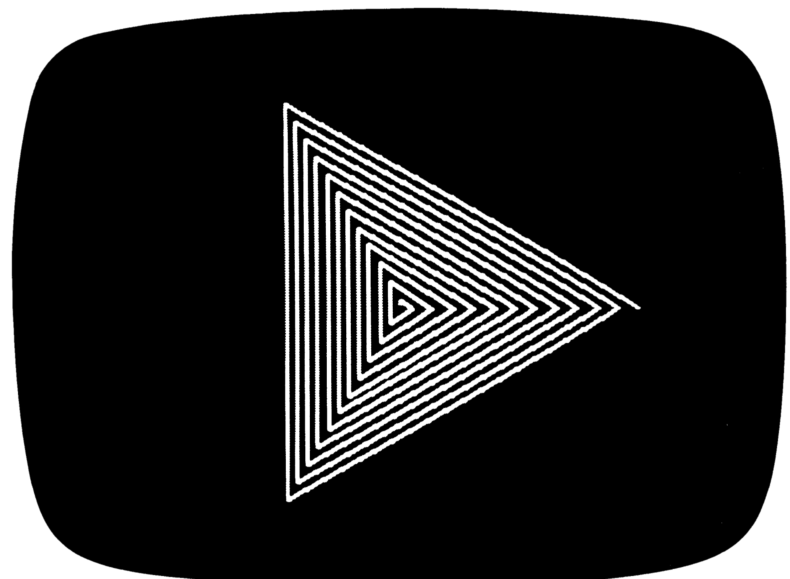


Figure 9-24. The spiral drawn by TRI.SPIRAL.2, with each corner 120 degrees

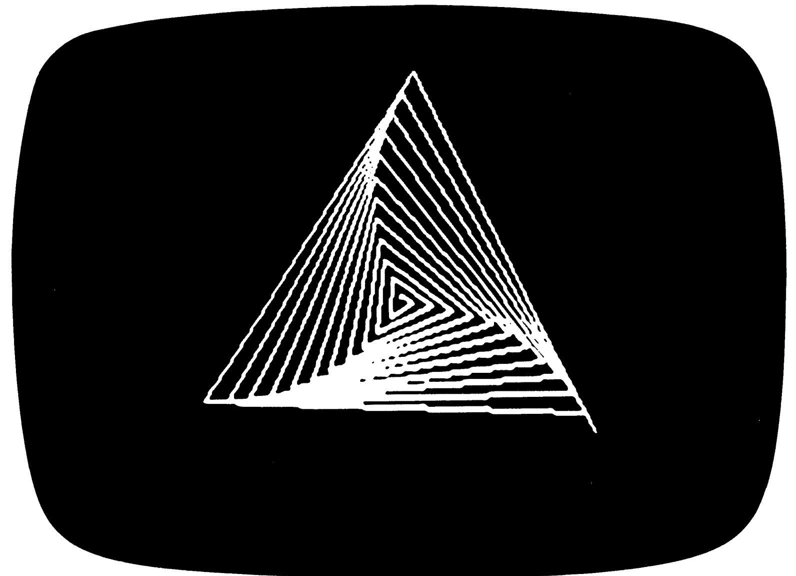


Figure 9-25. The spiral drawn by TRI.SPIRAL.2, with each corner 121 degrees

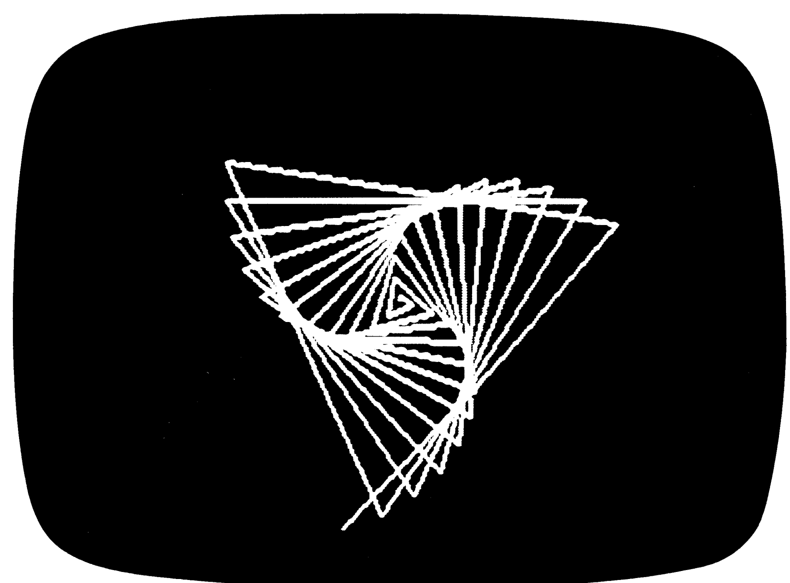


Figure 9-26. The spiral drawn by TRI.SPIRAL.2, with each corner 123 degrees

You might also want to develop spiral procedures similar to those produced by STAR.SPIRAL and STAR.RAINBOW that draw designs composed of figures other than five-pointed stars. For example, replacing the five-pointed stars with 10-pointed stars in STAR.SPIRAL gives the design shown in Figure 9-27. The use of polygonal shapes such as triangles and octagons might produce interesting spiral designs as well.

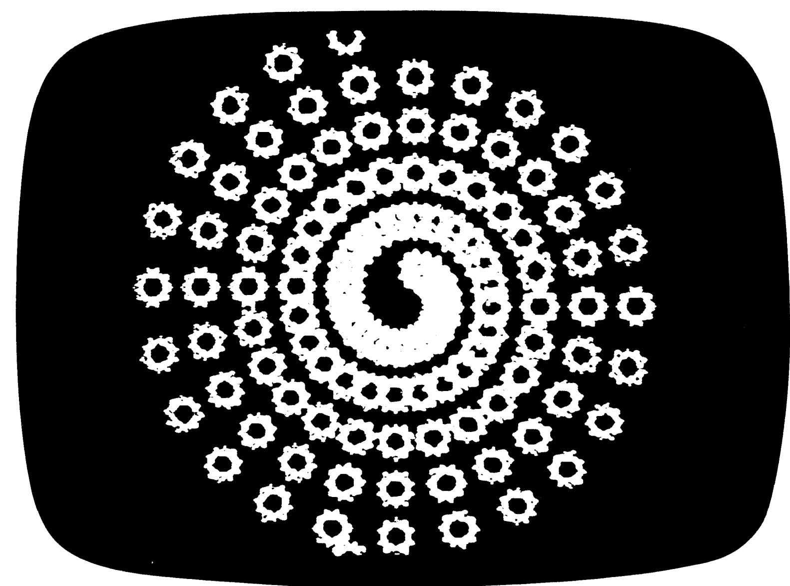


Figure 9-27. The spiral drawn by STAR.SPIRAL, with 10-pointed stars

Logo Activity Time

FRACTIONS, a game on your Logo Activity Disk, uses the STAR procedure to let you check visually whether two given fractions are equivalent—that is, whether they have the same reduced form. Load the FRACTIONS file into your computer from your Logo Activity Disk by typing:

```
READ "FRACTIONS
```

Now, let's see how FRACTIONS works.

When the file has been read into your workspace, call for the FRACTIONS procedure by typing:

```
FRACTIONS
```

When you press <RETURN>, your screen switches to the Draw mode with a SplitScreen format, a blue background, and a yellow pen color. In the text portion of the screen, you're asked to select numbers for A and B in the fraction A/B. Then, you're asked to select numbers for C and D in the fraction C/D. Once you've selected the four values, the STAR

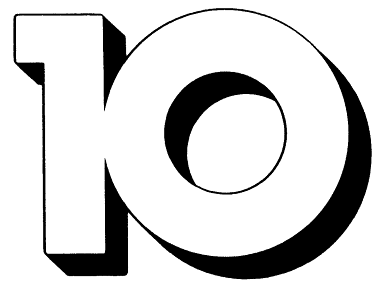
procedure draws on the screen, one at a time, the figures produced by the numbers you picked. If A/B and C/D are equivalent, a message in the text area tells you so and the two figures are identical. If A/B and C/D are not equivalent, a message in the text area tells you so and the two figures aren't identical. A message in the text area then asks if you want to play again. If you do, just type YES and you'll be asked for a new set of numbers for A, B, C, and D. If you don't want to play again, type NO (or anything else except YES) and you'll be returned to the Draw mode with a clear screen and the default background and pen colors.

The idea of FRACTIONS is for you to select numbers for A, B, C, and D which form equivalent fractions. Try to determine mathematically, before you enter the numbers by typing them on the keyboard, whether you think they are equivalent. Then, enter them and see if you're right. Remember, two fractions are equivalent only if they have the same reduced form.

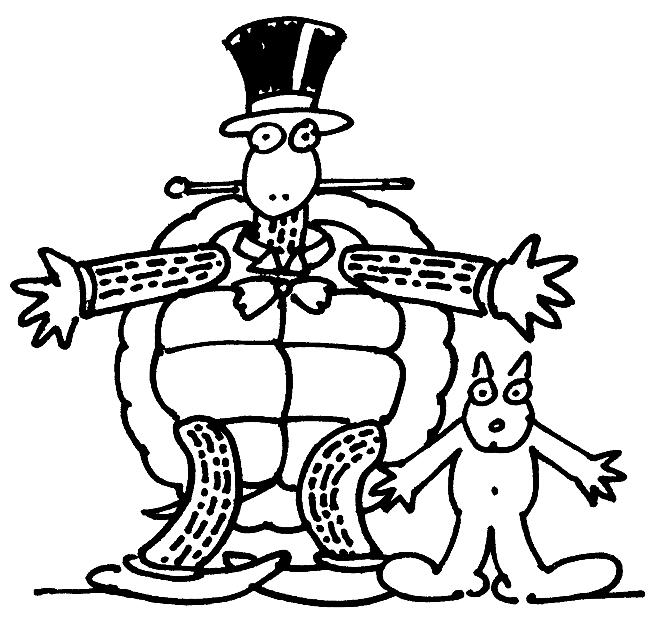
Some values of A, B, C, and D may result in figures too large to fit on one side of the screen, and may appear partly on one side and partly on the other.

New Commands Introduced In This Chapter

Command	*Abbreviation*
MAKE	
PRINTOUT NAMES	PO NAMES
ERASE NAMES	ER NAMES



Nonstop Procedures



Some of the procedures you've used so far have repeated an action a specific number of times by using the REPEAT command. But Logo also provides a way of making procedures repeat an action an unlimited number of times: recursion. Recursion is a little like perpetual motion; once it begins, it could go on forever. Fortunately, Logo also provides ways of ending recursion.

Recursion is easy to achieve. After you give a procedure a name, you use that name within the procedure as a subprocedure. Doing this results in endless repetition of the procedure because the computer reads the commands one at a time from the top of the procedure to the bottom. When the computer arrives at the name of the procedure at the point where it is used as a subprocedure, the computer returns to the top line to find out the definition of the command.

Recursion is especially handy in creating games. Use it to animate the turtle, or set it in endless motion, and you can then turn your attention elsewhere.

Let's see how recursion works with a simple procedure that draws the same square over and over. The following procedure, called KEEP.SQUARING, draws one square, then continues retracing the square using recursion:

```
TO KEEP.SQUARING
 FULLSCREEN BG 0
 FD 75
 RT 90
 KEEP.SQUARING
END
```

Let's see what KEEP.SQUARING does, one line at a time. First, it names the procedure. Then it sets the screen format to FullScreen and the background color to black. Next, it moves the turtle forward 75 steps, turns the turtle 90 degrees to the right, and then uses the procedure name, referring the computer back to the top line and starting the procedure all over again. When a procedure uses a procedure name as one of its commands like this, we say that the procedure "calls itself." It's as though the procedure were calling itself by name to summon help in doing some more work.

Since every repetition of KEEP.SQUARING results in another repetition, the procedure could repeat indefinitely and the turtle would continue to move around the square

until the procedure is somehow terminated. To verify it for yourself, enter KEEP.SQUARING. When you want to stop the procedure, use the <CTRL> G command.

Recursion is similar to a REPEAT statement, in that it results in repetition. But with recursion, you don't have to specify beforehand how many repetitions you want to occur. Let's look at another example. Suppose you want to use recursion to draw a square-spiral design like the one produced by the SQ.SPIRAL procedure in the last chapter. You could use the following recursive procedure called SQ.OUT:

```
TO SQ.OUT :DISTANCE
 FULLSCREEN BG 0
 FD :DISTANCE
 RT 90
 SQ.OUT :DISTANCE+3
END
```

SQ.OUT moves the turtle forward a distance represented by the variable :DISTANCE (which you specify when you run the procedure). Then, the turtle turns right 90 degrees. The procedure then calls itself by using its own name, but with the value for the variable :DISTANCE increased by 3. This causes the turtle to move forward by three additional steps after each turn, drawing a square-shaped spiral. If you use an initial value of 1 for :DISTANCE by typing SQ.OUT 1, the turtle will move one step to the first turn, four steps to the second turn (1+3=4), seven steps to the third turn (4+3=7), and so on. Type SQ.OUT 1 now to verify this for yourself.

More Things to Do

When you developed the SQ.SPIRAL procedure in the last chapter, you also developed a modified version, called SQ.SPIRAL.2, which used a variable name for the angle the turtle turns after drawing each side of the spiral. Rewrite the SQ.SPIRAL.2 procedure using recursion instead of the REPEAT statement, the same way you did with SQ.OUT. Call

this new procedure SQ.OUT.2. After you've written it, run it with a turn of 87 degrees, then 89 degrees, then 91 degrees, then 93 degrees, to verify that SQ.OUT.2 draws the same designs as SQ.SPIRAL.2.

Using Conditional Statements To Stop Recursion

The only problem with SQ.OUT so far is that it keeps going and going, making the spiral larger and larger, and you can only stop it with the <CTRL> G command. What's needed is a command that can be used inside the procedure itself to stop it automatically when the design gets to a specified size. That command is STOP. The STOP command can be used in conjunction with other commands that test for specified conditions, such as size. There are several such commands in Logo.

If-Then

Let's start with a fairly simple command that stops recursion: IF-THEN. IF-THEN is a conditional statement; it tells the computer what to do *if* a specified condition exists.

To use IF-THEN, type IF, followed by a description of the condition you want the procedure to check for. Then type THEN, followed by a description of the action you want the computer to take if the condition exists.

For example, suppose you want the procedure to check the turtle's X coordinates as it draws the square spiral and you want the procedure to stop when the X coordinate is greater than 100. As you may recall from Chapter 5, the expression for the X coordinate of the turtle is XCOR. So, you'd write:

```
IF XCOR > 100 THEN STOP
```

The > symbol means "is greater than." The < symbol means "is less than."

If you include the conditional statement in SQ.OUT just before the procedure calls itself again, the revised form of SQ.OUT looks as follows:

```
TO SQ.OUT :DISTANCE
 FULLSCREEN BG 0
 FD :DISTANCE
 RT 90
 IF XCOR > 100 THEN STOP
 SQ.OUT :DISTANCE + 3
END
```

You don't actually need the word THEN in the IF-THEN statement. You can simply type IF XCOR > 100 STOP and the computer understands what you mean. I tend to say to myself "If. . . then. . . ," so I always write it that way, as well. But you can do it whichever way you prefer.

When the computer reaches a conditional statement in the procedure, it checks to see if the stated condition exists. If it exists, the computer reads the rest of the line and does whatever it says to do. (In SQ.OUT, if the X coordinate is greater than 100, the computer stops the procedure.) If the stated condition doesn't exist, the computer doesn't even read the rest of the line; it simply continues to the next line of the procedure.

Run the revised SQ.OUT procedure to verify that the conditional command does, in fact, stop the procedure once the spiral becomes so big that the turtle's X coordinate is larger than 100.

True-False

Another way to obtain the same result is with the conditional commands TEST, IFTRUE (or IFT), and IFFALSE (or IFF). Type TEST first, followed by a space, and a condition that's always either true or false. IFTRUE (or IFT), IFFALSE

(or IFF), or both IFTRUE and IFFALSE are then used to tell the computer what action to take. When IFTRUE and IFFALSE occur in a procedure, they use the result of the most recent TEST command. So, they don't have to immediately follow the TEST; they can be separated from TEST by other commands. Furthermore, IFTRUE and IFFALSE should never be used in the same line of a procedure. If they are, and the first conditional is false, the computer will automatically skip the remainder of the line and not read the second conditional.

In SQ.OUT, we want to test the condition XCOR > 100 and we want the computer to stop the procedure when this condition is true. Using TEST and IFTRUE, you can obtain this result with the following commands:

```
TEST XCOR > 100
IFTRUE THEN STOP
```

The use of the word THEN is optional after IFTRUE and IFFALSE, just as it was with IF in the IF-THEN statement.

Compare the revised form of SQ.OUT, called SQ.OUT.3, which uses TEST, with the original form of the procedure SQ.OUT. Then run SQ.OUT.3 to verify that it gives the same result as SQ.OUT:

```
TO SQ.OUT.3 :DISTANCE
 FULLSCREEN BG 0
 FD :DISTANCE
 RT 90
 TEST XCOR > 100
 IFTRUE THEN STOP
 SQ.OUT.3 :DISTANCE + 3
END
```

More than One Condition

You can also use more than one conditional statement in the same procedure. For example, if you want the procedure to stop when either the X coordinate or the Y coordinate

becomes larger than 100, you can use these two IF-THEN conditionals:

```
IF XCOR > 100 THEN STOP
IF YCOR > 100 THEN STOP
```

Or, you can use these equivalent TEST commands:

```
TEST XCOR > 100
IFTRUE THEN STOP
TEST YCOR > 100
IFTRUE THEN STOP
```

Here is an alternative shorthand form that is used in Logo:

```
IF ANYOF XCOR > 100 YCOR > 100 THEN STOP
```

You could also type the same command:

```
TEST ANYOF XCOR > 100 YCOR > 100
IFTRUE THEN STOP
```

The expression ANYOF can be used after IF or TEST when you have two conditions and you want the computer to take a specified action if any of the conditions (either one or both) exist. Similarly, the expression ALLOF can be used after IF or TEST when there are two conditions to check and you want the computer to take a specified action only if both conditions exist. You can also use ANYOF and ALLOF after IF or TEST with more than two conditions. But when you do, the words ANYOF and ALLOF must be preceded by an opening parenthesis and the final condition must be followed by a space and a closing parenthesis. Following are several examples illustrating the use of ANYOF and ALLOF with two and more than two conditions. First, the single command using IF ANYOF:

```
IF ANYOF XCOR > 100 YCOR > 100 THEN STOP
```

And the equivalent pair of commands, using TEST ANYOF:

```
TEST ANYOF XCOR > 100 YCOR > 100
IFTRUE THEN STOP
```

Next, the single command, using IF ALLOF:

```
IF ALLOF XCOR > 100 YCOR > 100 THEN STOP
```

And the equivalent pair of commands, using TEST ALLOF:

```
TEST ALLOF XCOR > 100 YCOR > 100
IFTRUE THEN STOP
```

The single command, using IF ANYOF with multiple conditions:

```
IF (ANYOF XCOR > 100 XCOR < -100 YCOR
 > 100 YCOR < -100 ) THEN STOP
```

And the equivalent pair of commands, using TEST ANYOF with multiple conditions:

```
TEST (ANYOF XCOR > 100 XCOR < -100 YCOR
 > 100 YCOR < -100 )
IFTRUE THEN STOP
```

Finally, the single command, using IF ALLOF with multiple conditions:

```
IF (ALLOF XCOR < 50 XCOR > -50 YCOR
 < 50 YCOR > -50 ) THEN STOP
```

And the equivalent pair of commands, using TEST ALLOF with multiple conditions:

```
TEST (ALLOF XCOR < 50 XCOR > -50 YCOR
 < 50 YCOR > -50 )
IFTRUE THEN STOP
```

If-Then-Else

One other conditional statement you might use is IF-THEN-ELSE. Both THEN and ELSE are followed by actions the computer might take. If the condition following

IF exists, it takes the action following THEN. If the condition following IF doesn't exist, it takes the action following ELSE instead. An example of IF-THEN-ELSE is:

```
IF XCOR > 100 THEN PC 7 ELSE PC 5
```

This statement tells the computer to check the X coordinate of the turtle, XCOR. If the X coordinate is greater than 100, then the computer sets the turtle's pen color to yellow (number 7). If the X coordinate is not greater than 100, the computer sets the turtle's pen color to green (number 5).

Combining Recursion and Conditional Statements

Recursion and conditional statements can be used together in a variety of imaginative ways. For example, you can define a procedure that uses recursion and a conditional statement to draw larger and larger squares, clearing the screen and changing the pen color between successive squares, then stopping when the length of the sides becomes larger than 200. Since the largest pen color number is 15, you need a way of making sure the variable :PENCOLOR never gets larger than 15. You can do that by including the conditional statement:

```
IF :PENCOLOR=16 THEN MAKE "PENCOLOR 1
```

The entire procedure, called BIGGER.SQUARES, looks like this:

```
TO BIGGER.SQUARES :SIDE :PENCOLOR
 FULLSCREEN BG 0
 IF :SIDE > 200 THEN STOP
 IF :PENCOLOR=16 THEN MAKE "PENCOLOR 1
 CS
 PU SETXY (-:SIDE/2) (-:SIDE/2) PD PC :PENCOLOR
 REPEAT 4 [FD :SIDE RT 90]
 BIGGER.SQUARES :SIDE+10 :PENCOLOR+1
END
```

Run the procedure, using 10 as the starting length of the sides and white (number 1) as the starting pen color, by typing BIGGER.SQUARES 10 1.

More Things to Do

Write a procedure similar to BIGGER.SQUARES that erases each square after it's drawn by giving the pen an eraser (PC −1) or by using PENERASE, instead of erasing each successive square using CLEARSCREEN (or CS).

Instead of having the conditional statement stop the procedure, modify BIGGER.SQUARES so that the turtle redraws all of the squares in reverse order, starting with the largest and ending with the smallest, after the squares reach a certain size.

Instead of having the procedure draw squares around other squares, set up a procedure that draws successively larger squares starting at the left edge of the screen and moving to the right. Erase each previous square using the CLEARSCREEN command before the next square is drawn. To add a little more challenge, use a conditional statement that automatically starts a new square back at the left edge of the screen when the turtle nears the right edge of the screen.

Try these activities with figures other than squares just for practice in planning and writing procedures involving recursion and conditional statements.

Commands for Immediate Computer Response

Using what you've learned so far in this chapter about recursion and conditional statements, you can now put the turtle in uninterrupted motion. That's known as animation. Once the turtle has been put in motion, you can control its movement using two new commands: READCHARACTER (or RC) and RC?.

Let's start out with a simple animation procedure called MOVE. MOVE animates the turtle by making it move forward five steps, repeating the same five steps over and over by calling the procedure's own name recursively. MOVE doesn't contain any conditional statements to stop the turtle yet. So, when you want to end the procedure, press <CTRL> G.

```
TO MOVE
 FULLSCREEN BG 0 PU
 FD 5
 MOVE
END
```

Now, let's modify MOVE so that the turtle turns around and moves in the opposite direction when it gets too close to the edge of the screen. If it didn't turn, it would go off the edge and reappear on the opposite side. You can turn the turtle around at the edge of the screen by inserting the following conditional statement:

```
IF (ANYOF XCOR > 100 XCOR < -100 YCOR
 > 100 YCOR < -100 ) THEN RT 180
```

This conditional statement checks the turtle's X and Y coordinates to see if the X coordinate is greater than 100 (near the right edge of the screen), or if the X coordinate is less than −100 (near the left edge of the screen), or if the Y coordinate is greater than 100 (near the top edge of the screen), or if the Y coordinate is less than −100 (near the bottom edge of the screen). If any one of these four conditions is satisfied, the turtle is instructed to turn 180 degrees to the right, reversing its direction.

Using the conditional statement in MOVE makes it a new procedure called MOVE.2 that looks like this:

```
TO MOVE.2
 FULLSCREEN BG 0 PU
 IF (ANYOF XCOR > 100 XCOR < -100 YCOR
  > 100 YCOR < -100 ) THEN RT 180
 FD 5
 MOVE.2
END
```

If you point the turtle in any direction and run MOVE.2, it will move forward until it gets far enough away from the center of the screen for its X or Y coordinate to satisfy one of the four conditions of the conditional statement. When this happens, it turns around and moves in the opposite direction. Try it and see.

Now, let's improve on MOVE.2 by drawing a yellow square on the screen to indicate where the turtle will make its turns. You can do that by having a subprocedure (call it DRAW.SQUARE) draw the yellow square on the screen. You can also have a subprocedure called ANIMATE.TURTLE animate and turn the turtle essentially the same way MOVE.2 does. Have the new main procedure, called MOVE.3, set the screen format and the background color, then call the procedures DRAW.SQUARE and ANIMATE.TURTLE to draw the square and move the turtle around inside the square. These procedures look like this:

```
TO MOVE.3
 FULLSCREEN BG 0 PU
 DRAW.SQUARE
 ANIMATE.TURTLE
END
```

```
TO DRAW.SQUARE
 SETXY (−100) (−100) PC 7 PD
 SETY 100 SETX 100 SETY −100 SETX −100
 PU HOME
END

TO ANIMATE.TURTLE
 IF (ANYOF XCOR > 100 XCOR < −100 YCOR
  > 100 YCOR < −100 ) THEN RT 180
 FD 5
 ANIMATE.TURTLE
END
```

If you enter these procedures and run MOVE.3, you'll get the same results as with MOVE.2, except that the turtle appears to bounce back and forth between the top and bottom of the yellow square (assuming the turtle is facing straight up when the procedure begins), as in Figures 10-1, 10-2, and 10-3.

Figure 10-1. The screen at the start of MOVE.3

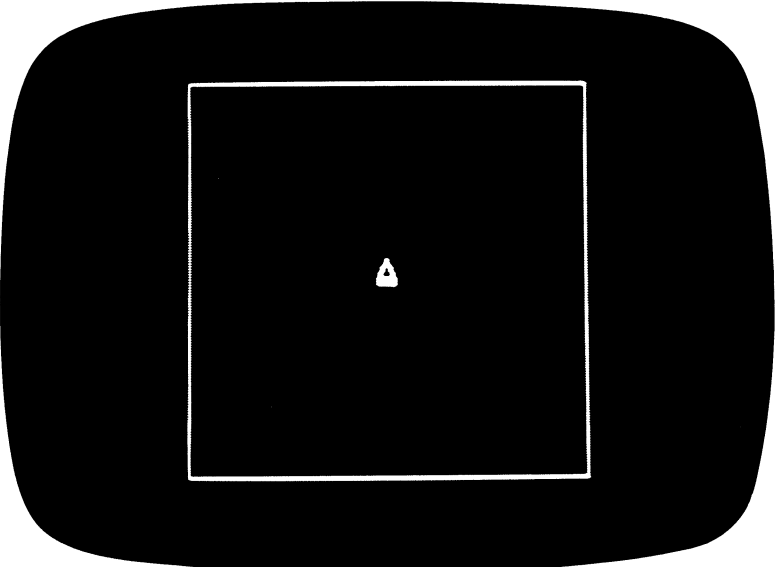


Figure 10-2. MOVE.3, as the turtle nears the top of the square

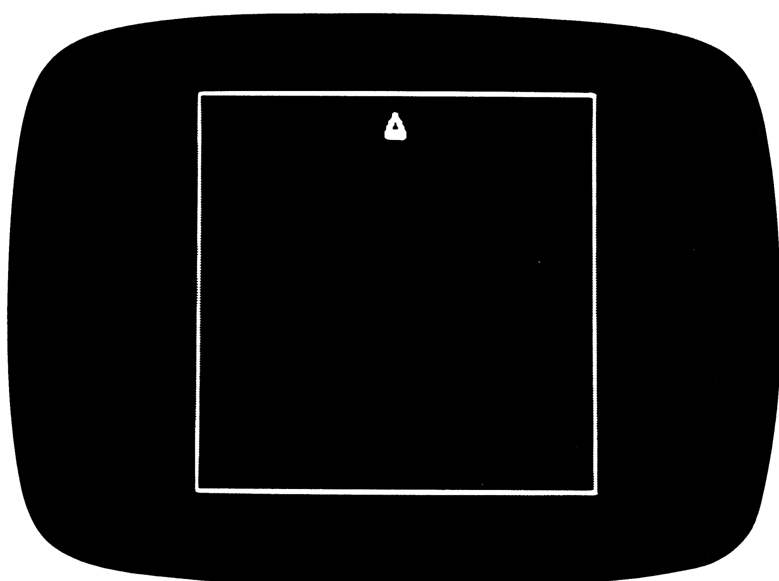
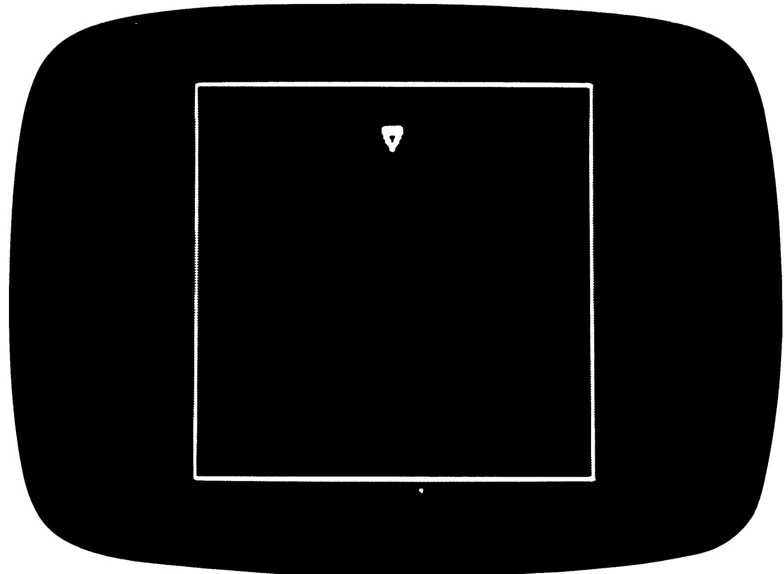


Figure 10-3. MOVE.3, after the turtle bounces back from the top of the square



Now, let's extend the animation procedure one final time so you can control the turtle's direction as it moves around inside the square. To do that, you'll need to use two new Logo commands, READCHARACTER (or RC) and RC?.

The RC? command asks the computer to check whether a key has been pressed on the keyboard. The computer answers "Yes" if a key has been pressed and "No" if a key has not been pressed. READCHARACTER (or RC) tells the computer to read the key that has been pressed or to wait if a key hasn't been pressed.

The RC? and RC commands are often used together in a conditional statement. If a key has been pressed, the RC command tells the computer which of several actions to take, depending on which key was pressed. The following statement is an example of this:

```
IF RC? THEN MAKE "KEY RC CHANGE
```

If no key has been pressed, then the condition RC? doesn't exist and the computer doesn't read the rest of the line. If a key has been pressed, the computer reads the rest of the line. It is then instructed to represent whichever key was pressed by the variable name "KEY; and to find the subprocedure named CHANGE and do what CHANGE tells it to do. CHANGE tells the computer what action to take.

The following are two procedures that put the turtle in motion. The first, ANIMATE.TURTLE, is a revised version of the procedure with the same name given earlier. This revised version contains the conditional statement that asks the computer to check and see if you've pressed a key. If you have, the computer reads which key it is and looks for instructions on its next course of action. The second procedure, CHANGE, tells the computer what to do if a key has been pressed. It sets the turtle's heading to 0 if the <U> key (for up) is pressed, to 90 if the <R> key (for right) is pressed, to 180 if <D> (for down) is pressed, or to 270 if <L> (for left) is pressed. If <E>

key is pressed, the computer ends the procedure and puts the screen back in Draw mode.

```
TO ANIMATE.TURTLE
 IF (ANYOF XCOR > 100 XCOR < -100 YCOR
  > 100 YCOR < -100 ) THEN RT 180
 IF RC? THEN MAKE "KEY RC CHANGE
 FD 5
 ANIMATE.TURTLE
END

TO CHANGE
 IF :KEY = "U THEN SETH 0
 IF :KEY = "R THEN SETH 90
 IF :KEY = "D THEN SETH 180
 IF :KEY = "L THEN SETH 270
 IF :KEY = "E THEN DRAW SPLITSCREEN TOPLEVEL
END
```

Instead of using the STOP command to stop the turtle's motion when the <E> key is pressed, the CHANGE subprocedure uses a new command, TOPLEVEL. STOP only stops the procedure in which it appears. If STOP is used in a subprocedure, as it is here, it only stops the subprocedure and the main procedure continues to run. TOPLEVEL, on the other hand, stops all of the procedures at once, no matter which subprocedure it appears in.

Edit ANIMATE.TURTLE to include this revision and enter the CHANGE procedure. Then run MOVE.3 once again. The turtle will continue to bounce off the walls of the yellow square, but you should now be able to change its direction as it's moving by pressing the <U>, <R>, <D>, and <L> keys. When you want to make the procedure stop, press <E>.

More Things to Do

You can modify MOVE.3 to have the turtle respond in different ways by adding new commands to the subprocedure CHANGE. For example, you could add the following command:

```
IF :KEY = "S THEN PD REPEAT 4 [FD 20 RT 90] PU
```

which will make the turtle lower its pen, draw a square with sides 20 steps long, then raise its pen again every time you press the <S> (for square) key. Try adding this extra command to CHANGE and see how it works.

Now, add additional commands to CHANGE that make the turtle draw a triangle whenever you press the <T> key and draw a star with five points whenever you press the <5> key. (Remember to make the turtle lower its pen before drawing the figures and raise it afterward.) Using these additional commands, you should be able to create screen displays similar to those illustrated in Figures 10-4 and 10-5.

Figure 10-4. A design drawn by CHANGE

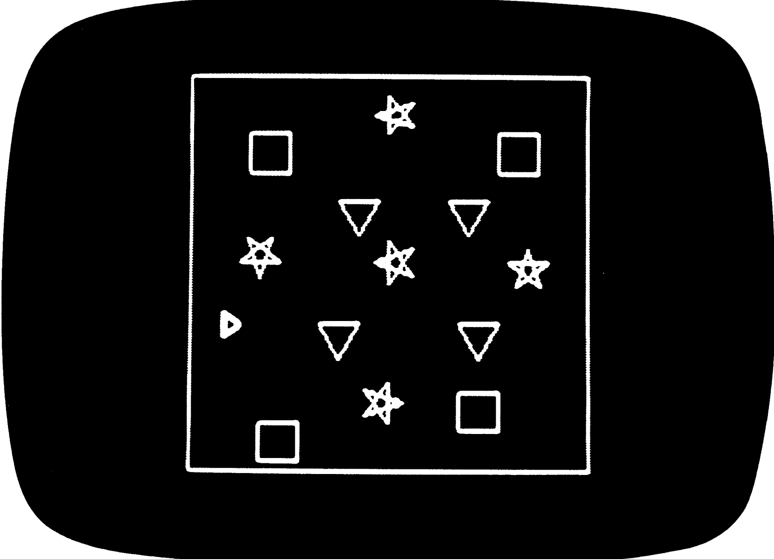
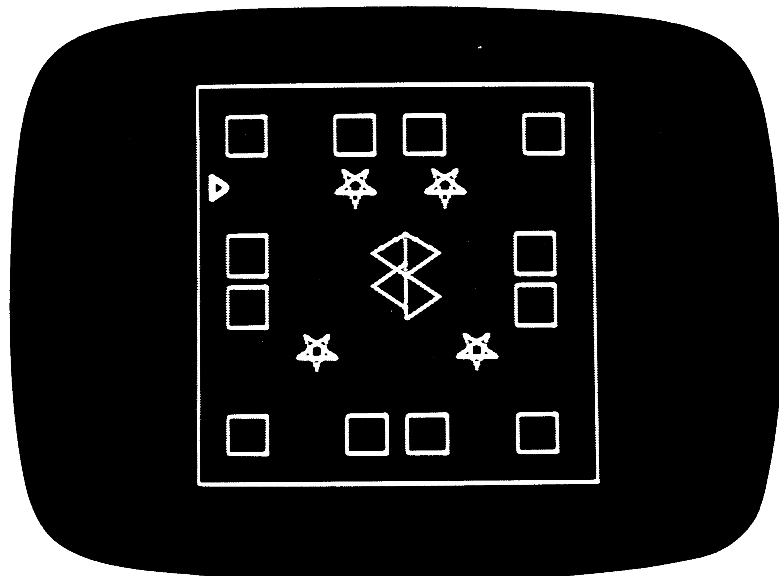


Figure 10-5. Another design drawn by CHANGE



Try to think of some other actions you'd like the turtle to take and add them to CHANGE as well. You can even add commands that change the background color of the screen or the color of the turtle's pen.

Logo Activity Time

SPACETURTLE, a game on your Logo Activity Disk, is an extension of the procedure MOVE.3 developed in the last section. It uses recursion to give the turtle animated movement, conditional statements to make the turtle bounce off a screen border, and the commands RC? and RC to make the turtle respond immediately to your commands, depending on which keys on the keyboard you press. Read this activity into the computer from your Logo Activity Disk, using the command:

```
READ "SPACETURTLE
```

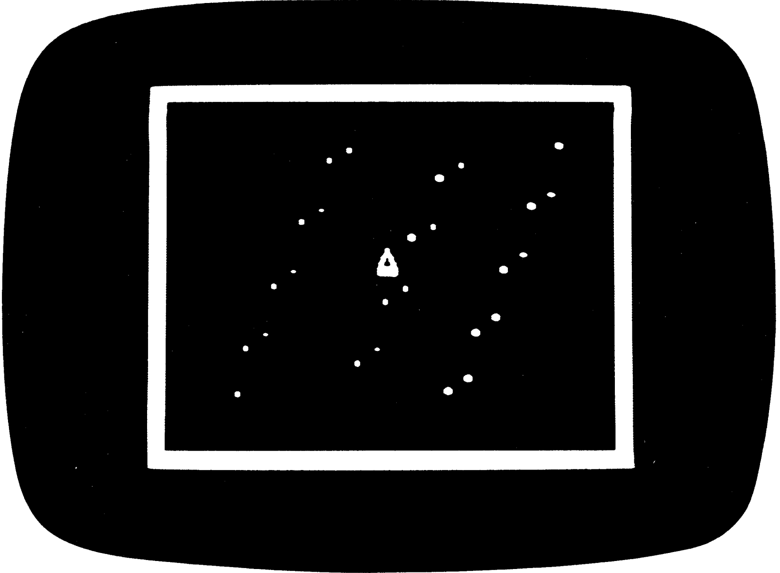
The main procedure in SPACETURTLE has the same name as the file it's in. So, once the file has been read into your computer, just type:

```
SPACETURTLE
```

In SPACETURTLE, you are a spaceship commander and the turtle is your spaceship. The game begins with the screen in the FullScreen format with a blue background and a yellow rectangle enclosing the part of the screen serving as the sector of space that your spaceship has been ordered to patrol.

Inside this sector of space are 26 small yellow asteroids and your spaceship, which appears at the center and immediately begins moving up toward the top of the screen. As in MOVE.3, your spaceship has been programmed to bounce off the sides of the yellow rectangle. You can make it move up by pressing <U>, right by pressing <R>, down by pressing <D>, and left by pressing <L>. When the game begins, your screen should look like Figure 10-6.

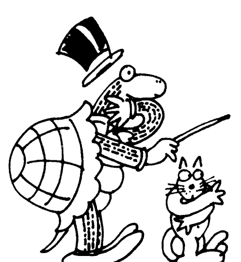
Figure 10-6. The screen at the start of SPACETURTLE



Your orders are to clear the sector of the 26 asteroids. Arming your ship is a phaser, which can destroy the asteroids. Pressing the <S> key shoots your phaser. When you press <S>, your ship stops moving and a black ray sweeps a short distance in front of it. When the ray clears, any asteroid that it covered is gone and your ship resumes its movement. (The ray is drawn by a second, invisible turtle that moves forward and backward a large number of times, with a right turn between each move forward and backward. After the ray is drawn, the action repeats with the pen color changed to PENCOLOR−1 to erase both the black color and any yellow color representing an asteroid. You don't see the turtle moving back and forth, drawing and erasing the ray, because I've used an invisible Logo sprite to do the drawing and erasing in order to improve the way the activity looks. You'll learn what sprites are and how to use them in the next chapter. Once you learn how to use sprites, you could develop or modify this activity yourself.)

When all the asteroids have been destroyed, press the <E> key to end the game. The display tells you how many phaser shots you needed to destroy all 26 asteroids. You should be able to clear the asteroids with fewer than 26 shots by positioning your spaceship close to several of them before shooting. If you want to play again, type SPACETURTLE. Otherwise, simply type DRAW and continue with the Logo activities in the next chapter.

SPACETURTLE uses little more than what's covered in this chapter and should give you an idea of the fairly sophisticated projects you should now be able to develop on your own. After you've finished with the next chapter, you'll know everything that went into the development of this activity and should be ready to start experimenting (if you haven't already) with creating activities of this sort yourself.



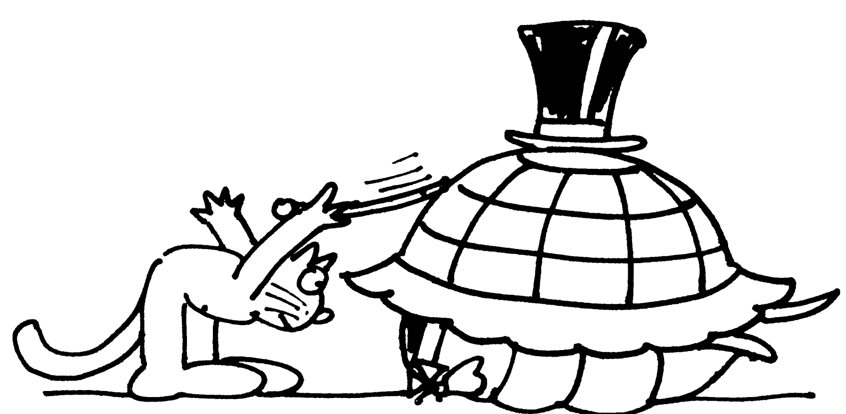
Turtle Trap: As I mentioned before, this activity uses an invisible Logo "sprite" (you'll learn more about sprites in the next chapter) to draw and erase the black ray when you press the <S> key to shoot your phaser. As you'll see in the next chapter, using the turtle as a spaceship and a sprite as the spaceship's phaser requires the procedures in SPACETURTLE to continually switch back and forth between telling the turtle and the sprite what to do.

If you end the game by pressing <E>, communication with the sprite ends automatically and you are ready to continue your Logo work with the turtle. If, however, you end the activity in mid-run by pressing <CTRL> G, it's possible you may interrupt the game while the procedure is addressing the sprite rather than the turtle. If that happens, any commands you give will be directed to the invisible sprite and the turtle will not respond. If the turtle isn't responding to your commands, simply type TELL 0 and press <RETURN>. The number 0 refers to the regular Logo turtle and the command tells the computer you want to talk to the turtle rather than the sprite. An alternative is to type GOODBYE, since this has the same effect as turning the computer off and on and loading Logo again. If you type GOODBYE, however, you'll lose any procedures you currently have in your workspace. So, using the command TELL 0 is preferable.

New Commands Introduced In This Chapter

Command	*Abbreviation*
IF-THEN	
IF-THEN-ELSE	
TEST	
IFTRUE	IFT
IFFALSE	IFF
ANYOF	
ALLOF	
READCHARACTER	RC
RC?	
STOP	
TOPLEVEL	
TELL	

11 Looking Into Sprites



The turtle isn't the only "object to think with" in Commodore 64 Logo. There are eight objects, called sprites, you can use just as you've been using the turtle. The turtle, in fact, is one of these eight sprites.

Sprites can take the shape of a number of common objects, from a butterfly to a truck. And you can have up to eight of them on the screen at one time.

Sprites: What They Are and How to Use Them

Commodore 64 Logo has eight sprites, including the triangular turtle. The sprites are numbered 0 through 7.

When you first enter Logo, you're automatically in control of Sprite 0, the triangular turtle. You can switch control to any other sprite by typing TELL, a space, and the number of the sprite you want to command. For example, suppose you want to see and command Sprite 1. First, put the screen in the Draw mode and hide the turtle by typing DRAW HT. Now, tell the computer you want to control Sprite 1 by typing TELL 1. Any commands you now give will be directed to Sprite 1 until you use the TELL command again, type GOODBYE, or leave and then reenter Logo.

Sprites 1 through 7 are hidden and have their pens up when you first enter Logo. So, you need to type SHOWTURTLE (or ST) if you want to make one of them appear and PENDOWN (or PD) if you want one to draw. Go ahead and tell Sprite 1 to show itself and put its pen down.

As you can see, Sprite 1 is a rectangular block. That's the shape it and every other sprite is automatically given by Logo. The rectangle is called its default shape.

Now, just for fun, try moving Sprite 1 around the screen, drawing pictures with it just as you would with the triangular turtle. Any commands you use with the triangular turtle can also be used with any of the other sprites. Try changing Sprite 1's pen color to yellow, for example, and then use a REPEAT statement to make it draw a square.

The sprite you're controlling at any particular time is referred to as your current sprite. Since you're currently controlling Sprite 1, Sprite 1 is your current sprite.

Using Logo's Predefined Sprite Shapes

As long as the sprites look like rectangles, they're not very useful for creating animated drawings or activities. You need to be able to give them shapes of your own choosing. As mentioned earlier, you can do this by using one of the many predefined sprite shapes that are available on the Logo Utilities Disk that comes with the Commodore 64 Logo package.

Loading and Viewing Shapes

The Logo Utilities Disk that comes with the Logo package contains five ready-to-use sets of shapes for Sprites 1 through 7. These sets are contained in files with the names ANIMALS, VEHICLES, SHAPES, RUNNER, and ASSORTED. The shapes contained in these files are shown in table form in Figure 11-1.

Figure 11-1. The shapes in the ANIMALS, VEHICLES, SHAPES, RUNNER, and ASSORTED files

Animals

Number	*Name*	*What it looks like*
1	DINOSAUR	
2	KANGAROO	
3	BUG	
4	DOLPHIN	
5	HORSE	
6	CAT	
7	BUTTERFLY	

(continued)

Figure 11-1. The shapes in the ANIMALS, VEHICLES, SHAPES, RUNNER, and ASSORTED files (continued)

Vehicles

Number	*Name*	*What it looks like*
1	TRUCK	
2	CAR	
3	BICYCLE	
4	SUBMARINE	
5	AIRPLANE	
6	BOAT	
7	BALLOON	

Shapes

Number	*Name*	*What it looks like*
1	FRAME	
2	BBALL (Big Ball)	
3	SBALL (Small Ball)	
4	SQUARE	
5	TRIANGLE	
6	HEART	
7	BOX	

(continued)

Figure 11-1. The shapes in the ANIMALS, VEHICLES, SHAPES, RUNNER, and ASSORTED files (continued)

Runner

Number	*Name*	*What it looks like*
1		
2		
3		
4		
5		
6		
7		

Assorted

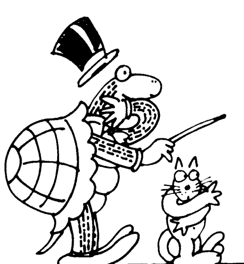
Number	*Name*	*What it looks like*
1	TARGET	
2	TRUCK	MILK
3	ROCKET	
4	BALLOON	
5	BOW	
6	ARROW	
7	MAN	

To see how these files are used and what the shapes look like, load the set of vehicle shapes by inserting the Utilities Disk into your disk drive and typing READ "VEHICLES on your keyboard. When the red light on your disk drive goes off, Shapes 1 through 7 in the VEHICLES file will be in your workspace, with each shape assigned to its correspondingly numbered sprite.

Hide the triangular turtle by typing TELL 0 and HIDETURTLE (or HT), and use the commands TELL and SHOWTURTLE (or ST) to make Sprites 1 through 7 appear on the screen one at a time so you can see what they look like. Make sure you hide each sprite before switching communication to the next one so they won't be superimposed, one on top of another, on the screen.

An alternate way to view all eight current sprite shapes is with the following DISPLAY.SPRITES procedure. DISPLAY.SPRITES shows all eight sprites on the display screen at the same time. Sprites 0 through 3 are displayed at the top of the screen from left to right, while Sprites 4 through 7 are shown at the bottom of the screen from left to right, as in Figure 11-2.

```
TO DISPLAY.SPRITES
 FULLSCREEN BG 0
 MAKE "X (-120)
 MAKE "NUMBER 0
 REPEAT 4 [TELL :NUMBER PU SETXY (:X) 75 ST
  MAKE "X :X+80 MAKE "NUMBER :NUMBER+1]
  MAKE "X (-120)
 REPEAT 4 [TELL :NUMBER PU SETXY (:X) (-75) ST
  MAKE "X :X+80 MAKE "NUMBER :NUMBER+1]
  TELL 0
END
```

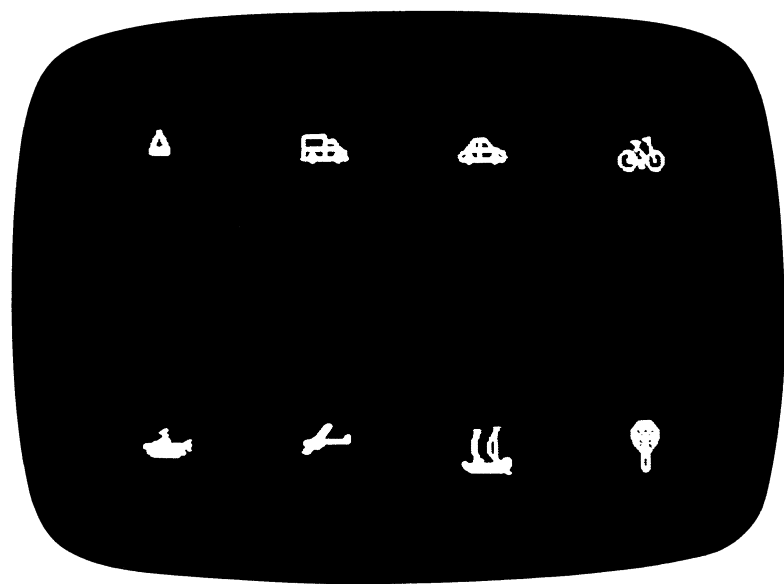


Turtle Trap: If you use DISPLAY.SPRITES and then try to clear the screen by typing DRAW, you'll find that only the turtle (Sprite 0) moves back to the Home position. The other sprites don't move because DISPLAY.SPRITES ends with TELL 0, making the turtle the current sprite. Even if you switch to the NoDraw mode and then back to the Draw mode again, only the turtle will respond. Sprites 1 through 7 remain fixed on the screen. To send Sprites 1 through 7 back to the Home position and hide them, you need to address them one at a time with the TELL command. That's exactly what ERASE.SPRITES is designed to do.

When you have finished looking at the sprites, you can use the following ERASE.SPRITES procedure to move all of the sprites back to the center of the screen, hide all except the triangular turtle, and make the turtle the current sprite.

```
TO ERASE.SPRITES
 MAKE "NUMBER 0
 REPEAT 8 [TELL :NUMBER PU HT HOME MAKE
  "NUMBER :NUMBER + 1]
 TELL 0 ST PD SPLITSCREEN BG 11
END
```

Figure 11-2. Sprites displayed by DISPLAY.-SPRITES



More Things to Do

Load the other four shape files into the computer's memory, one at a time, from your Utilities Disk. For each file, use DISPLAY.SPRITES to view all the shapes at the same time and ERASE.SPRITES to remove all but the turtle from the screen. Remember that whenever a new shape file is read in, it automatically replaces the shapes from the previous file. The shapes from these four files appear on your screen, one file at a time.

Load the set of vehicle shapes back into your workspace and display them again using DISPLAY.SPRITES. Then, type TELL 2 PC 7 and notice that Sprite 2, which has the car shape, is now yellow. Just as with the turtle, Sprites 1 through 7 always appear in the current pen color no matter which one is on the screen. Use this method to change some of the other vehicle shapes to other colors.

See if you can modify DISPLAY.SPRITES so that all the shapes are drawn in a color of your choice. Then, try to modify the procedure again so that the eight sprite shapes appear in eight different colors.

Assigning Pre-Defined Shapes to Sprites

When you load any sprite shape file but RUNNER into the computer from the Utilities Disk, two things happen. First, a set of seven shapes is established in the computer's workspace, with each of the shapes represented both by a number and a corresponding name. (For example, Shape 1 in the VEHICLES file is a truck, so you can use either Shape 1 or :TRUCK to refer to this shape when the VEHICLES file is in your workspace. As you'll see later in this chapter, you can use either a shape number or a shape name to assign a particular shape to a specified sprite.) Second, Shapes 1 through 7 are automatically assigned to Sprites 1 through 7,

with each shape assigned to its correspondingly numbered sprite. That's why, when you loaded the VEHICLES file, Sprite 1 was assigned Shape 1, a truck; Sprite 2 was assigned Shape 2, a car; and so on.

The shapes in the RUNNER file don't have names, so you can refer to them only by number. But, like the other four files, the RUNNER shapes are automatically assigned to the correspondingly numbered sprites when you load the RUNNER file.

Once these shapes are transferred from the Utilities Disk into the computer's memory and assigned a sprite number, however, you can reassign them using the command SETSHAPE, so that any of the sprites can have any of the shapes in the file. In fact, using SETSHAPE, you can even assign the same shape to more than one sprite at the same time. Let's see how shape assignment is done, using the animal shapes. Load the animal shapes into your workspace by typing READ "ANIMALS on your keyboard.

Loading the ANIMALS file automatically pairs Sprites 1 through 7 with Shapes 1 through 7, as follows:

Sprite	*Shape*
1	1 DINOSAUR
2	2 KANGAROO
3	3 BUG
4	4 DOLPHIN
5	5 HORSE
6	6 CAT
7	7 BUTTERFLY

Suppose, however, that you want to develop a country scene in which two butterflies serenely float through the air. You already have one butterfly shape, but you want to give another sprite the butterfly shape, too. Let's make Sprite 6 the other butterfly shape by using the TELL command to make it

the current sprite, and then instructing Sprite 6 to take the butterfly shape by using the SETSHAPE command with either the desired shape number (7) or the desired shape name (:BUTTERFLY). The command to do that is:

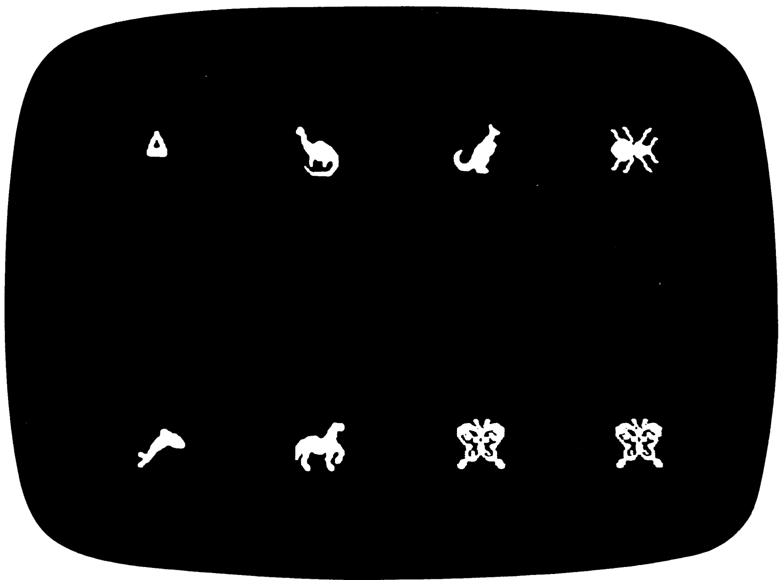
```
TELL 6 SETSHAPE 7
```

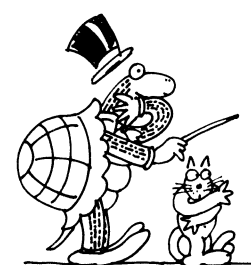
You could also type the same command this way:

```
TELL 6 SETSHAPE :BUTTERFLY
```

Now, both Sprite 6 and Sprite 7 have Shape 7, the butterfly shape, and you can use them as the two butterflies in your country scene. Use DISPLAY.SPRITES to verify for yourself that both Sprite 6 and Sprite 7 now have Shape 7, the butterfly shape, as in Figure 11-3.

Figure 11-3. Sprite 6 and Sprite 7 as butterflies





Turtle Trap: Once you assign a sprite a new shape, whether by name or number, that sprite takes the same shape number from any shape file you might load. For example, if you now load the VEHICLES file, both Sprite 6 and Sprite 7 will have Shape 7, a balloon. Try it for yourself by loading the VEHICLES file and then using DISPLAY.SPRITES again. If you want to give Sprite 6 its original Shape 6, you have to reassign Shape 6 to it by using the command TELL 6 SETSHAPE 6.

More Things to Do

Use SETSHAPE to give several different sprites the same shape, but give each a different color. For example, using animal shapes, make Sprites 1 through 3 dinosaurs of different colors and Sprites 4 through 7 kangaroos of different colors. Use DISPLAY.SPRITES to see the effects of these shape and color assignments.

Finding Out the Number and Shape Of the Current Sprite

Since you could have several different sprites showing on the screen at the same time and you could have several different sprites with the same shape, it's helpful to have a way of determining which sprite is current and what shape number is assigned to it. The commands that give this information are WHO and SHAPE.

If you type WHO, the screen will display the message RESULT:, a space, and the number of the current sprite. If you type SHAPE, the screen will display RESULT:, a space, and the shape number that is assigned to the current sprite. For example, suppose you've just assigned Shape 7 to Sprite 6 using the command TELL 6 SETSHAPE 7. If you type WHO, the message RESULT: 6 will appear on the screen. That tells you Sprite 6 is the current sprite. If you type SHAPE, the message RESULT: 7 will appear on the screen. That tells you Sprite 6 currently has Shape 7.

Some Sample Animations

Now that you've seen how to alter a sprite's shape and color, let's put these capabilities to use by developing an animated picture using vehicle shapes. Read the VEHICLES file into your workspace, if it isn't already there, by inserting the Utilities Disk into the disk drive and then typing READ "VEHICLES.

Let's begin with a simple picture consisting of a cyan (color number 3) balloon floating across a light gray (color number 15) screen from left to right. You can obtain this effect using the following two procedures. The FLY.BALLOON procedure calls on the ERASE.SPRITES procedure given earlier, so make sure ERASE.SPRITES is in your workspace.

```
TO FLY.BALLOON
 DRAW ERASE.SPRITES HT
 FULLSCREEN BG 15
 TELL 7
 SETXY (-120) 100
 PC 3 SETHEADING 90 ST
 ANIMATE.BALLOON
END

TO ANIMATE.BALLOON
 FD 2
 ANIMATE.BALLOON
END
```

FLY.BALLOON is the main procedure. It erases all sprites from the screen, sets the screen in the FullScreen format with a gray background, gives the current sprite the balloon shape, and makes it cyan-colored. The procedure then sets the balloon's starting position on the screen and the direction (heading) in which it will move. The ANIMATE.BALLOON subprocedure uses recursion to set the balloon in motion indefinitely at a gentle two steps per repetition. Enter these procedures and run FLY.BALLOON to see the effect they produce. When you want to stop FLY.BALLOON, press

<CTRL> G and type ERASE.SPRITES to erase all sprites except the triangular turtle from the screen.

Suppose, now, that in addition to the balloon floating at the top of the screen from left to right, you also want to have a blue submarine moving from right to left at the bottom of the screen. You can obtain this more interesting effect by using a main procedure that sets the screen format and background color, subprocedures that set each sprite's position and heading, and a final subprocedure that alternately moves each of the two sprites and recursively calls itself to keep them moving indefinitely. The main procedure MOVE.BAL.SUB, together with its subprocedures SET.BAL, SET.SUB, and ANIMATE.BAL.SUB, follow:

```
TO MOVE.BAL.SUB
 DRAW ERASE.SPRITES HT
 FULLSCREEN BG 15
 SET.BAL
 SET.SUB
 ANIMATE.BAL.SUB
END

TO SET.BAL
 TELL 7
 SETXY (−120) 100
 PC 3 SETHEADING 90 ST
END

TO SET.SUB
 TELL 4
 SETXY 120 (−100)
 PC 6 SETHEADING (−90) ST
END

TO ANIMATE.BAL.SUB
 TELL 7 FD 2
 TELL 4 FD 4
 ANIMATE.BAL.SUB
END
```

Type in and then run the MOVE.BAL.SUB procedure. When you run it, notice that the submarine appears to be traveling faster than the balloon. The difference in speed is produced by making the sprite with the submarine shape move a larger distance on each repetition than the sprite with the balloon shape (four steps as opposed to two steps) in the subprocedure ANIMATE.BAL.SUB.

Press <CTRL> G and type ERASE.SPRITES when you've finished looking at this animated display.

You can use this same technique of one main procedure calling several subprocedures to animate as many sprites at the same time as you want. You can even have the main procedure call subprocedures that use sprites to draw pictures on the screen, either before the animation begins or during the animation.

We'll end this first example of a sprite-animation activity by refining the balloon and submarine picture to include a boat. And, to make the picture more realistic, before the animation begins, let's use a hidden turtle to color in the bottom of the display screen blue to serve as a sea, draw an island rising out of the sea, and add a sun shining in the sky. With these added touches of realism, the submarine will appear to be moving beneath the water and the boat will appear to be sailing on top of the water.

Let's call the main procedure SEA.SCENE. Enter all of the following procedures and then run SEA.SCENE to see the sort of animation project you can now develop using Commodore 64 Logo's sprite capability. Figure 11-4 shows how the screen should look when the animation begins.

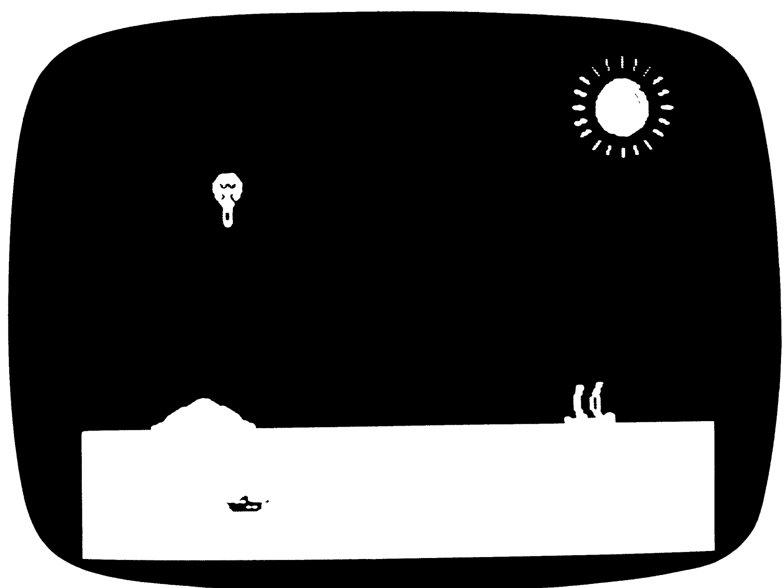


Figure 11-4. The screen at the start of SEA.SCENE

```
TO SEA.SCENE
 DRAW ERASE.SPRITES HT
 FULLSCREEN BG 14
 ISLAND
 WATER
 SUN
 SET.BALLOON
 SET.BOAT
 SET.SUBMARINE
 ANIMATE.B.B.S
END

TO ISLAND
 TELL 0 HT PU PC 15
 SETXY (−100) (−68) SETH 0 PD
 MAKE "L 25
 REPEAT 2 [LT 90 FD :L BK 2* :L FD :L RT 90 FD 1
  MAKE "L :L−3]
 MAKE "L :L+1
 REPEAT 4 [LT 90 FD :L BK 2* :L FD :L RT 90 FD 1
  MAKE "L :L−1]
```

```
  MAKE "L :L+1
  REPEAT 8 [LT 90 FD :L BK 2*:L FD :L RT 90 FD 1
   MAKE "L :L-2]
  PU HOME
END

TO WATER
  MAKE "Y (-70)
  TELL 0 PU HT PC 6 SETH 90
  PU SETXY (-160) (-70) PD
  REPEAT 60 [FD 320 RT 180 MAKE "Y :Y-1 SETY (:Y)]
  PU HOME
END

TO SUN
  TELL 0 HT PU PC 7
  SETXY 100 85 SETH 0 PD
  MAKE "S 4
  REPEAT 20 [REPEAT 20 [FD :S RT 18] MAKE "S :S-0.2]
  PU SETXY 112.626 87 SETH 0 PD
  REPEAT 20 [PU FD 20 PD FD 5 PU BK 25 RT 18]
  PU HOME
END

TO SET.BALLOON
  TELL 7 HT PC 3
  SETXY (-120) 50 ST
  SETH 90 ST
END

TO SET.BOAT
  TELL 6 HT PC 2
  SETXY 150 (-54)
  SETH (-90) ST
END

TO SET.SUBMARINE
  TELL 4 HT PC 1
  SETXY 125 (-100)
  SETH (-90) ST
END
```

```
TO ANIMATE.B.B.S
 TELL 7 FD. 2
 TELL 6 FD 4
 TELL 4 FD 3
 ANIMATE.B.B.S
END
```

Notice, when you run SEA.SCENE, that the boat appears to pass in front of the island. That's because of what's known as a display hierarchy that's built into Logo. This hierarchy dictates that a sprite always appears to move in front of, or takes precedence over, a screen drawing, and a lower-numbered sprite always appears to move in front of, or take precedence over, a higher-numbered sprite.

When you've finished looking at the animation, type <CTRL> G to stop the procedure.

A Quicker Way of Drawing Background Scenes

A procedure that begins with the turtle drawing a background scene, as SEA.SCENE does, takes a needlessly long time to complete. You can get to the animation more quickly by saving the entire background scene as a screen picture on the Work Disk. Then, when you want the picture drawn, you can have the main procedure simply read the picture into the computer from the disk before positioning and animating the sprites.

You can save the background scene, which is created by the ISLAND, WATER, and SUN procedures, by putting your Work Disk in the disk drive and typing:

```
SAVEPICT "SEA.SCENE
```

This creates the files SEA.SCENE.PIC1 and SEA.SCENE.PIC2 on your Work Disk.

To illustrate how much faster the display is produced

this way, rewrite the SEA.SCENE procedure to call on the picture that's stored on your Work Disk. In the following listing of the revised main procedure, you can see the changes to be made. Notice that the main procedure is now set up so that when it's run, it will read in the background picture from the Work Disk before positioning and animating the sprites.

```
TO SEA.SCENE.2
 DRAW ERASE.SPRITES HT
 FULLSCREEN
 READPICT "SEA.SCENE
 SET.BALLOON
 SET.BOAT
 SET.SUBMARINE
 ANIMATE.B.B.S
END
```

Self-Starting Files

In order to make this display a little bit more sophisticated, you can design a file to be self-starting—that is, set it up so that the main procedure begins running automatically as soon as the file is in the computer. You don't need to type the main procedure's name, as with other activities. You can do this by including the built-in Commodore 64 Logo variable name STARTUP as part of the file.

To make the file self-starting, use the assignment command MAKE to give the special variable name STARTUP a value. In this case, the value isn't a number, but a word, SEA.SCENE.2—the name of the file's main procedure. So, type:

```
MAKE "STARTUP [SEA.SCENE.2]
```

Notice that the procedure name must be enclosed within brackets.

If you want to save the procedures that form the file, make sure the variable STARTUP, together with its assigned value, is in your workspace. That way, it's saved in the file

with the procedures. Logo is designed so that when a file is read in from a disk, the computer automatically checks to see if the file contains the variable STARTUP. If it does, the computer reads the value of STARTUP. If the value is a procedure name, it automatically runs that procedure.

So, after you've set the variable STARTUP with the MAKE statement, save the procedures from this chapter by typing:

```
SAVE "CHAPTER.11
```

Let's see the effect of including STARTUP in the file in action. First, make sure the vehicle shapes are in your workspace. Then, place your Work Disk in the disk drive and type READ "CHAPTER.11. When you want to stop the display, press <CTRL> G and type ERASE.SPRITES. Once the file is read in, of course, you can keep running the procedures over and over by simply typing the name of the main procedure, SEA.SCENE.2, as long as the Work Disk is in your disk drive. The Work Disk has to be in the disk drive because the background picture is stored on it.

More Things to Do

The FLY.BALLOON and ANIMATE.BALLOON procedures were written to make the sprite with the balloon shape float straight across the screen from left to right at a constant height. You can make the scene a bit more realistic by having the balloon rise and fall as it moves across the screen, as if it were encountering air currents. You can achieve this up and down motion for the balloon in two different ways.

The two main procedures are FLY.BALLOON.2 and FLY.-BALLOON.3, with subprocedures ANIMATE.BALLOON.2 and ANIMATE.BALLOON.3. Look over the following listings of these procedures to see how they differ. Then, try them both to compare the effects they produce. I think FLY.-BALLOON.3 and ANIMATE.BALLOON.3 produce a smoother motion. See if you agree. Can you think of a third

way to produce this up and down motion? If you can, try writing out the corresponding procedures and running them to see if they work.

If you can't think of a completely different approach, try modifying the ANIMATE.BALLOON.2 subprocedure to make the balloon's motion smoother. Experiment with different numbers of repetitions and different numbers of steps within each REPEAT statement to see what effects these changes have.

```
TO FLY.BALLOON.2
 DRAW ERASE.SPRITES HT
 FULLSCREEN BG 15
 TELL 7
 SETXY (−120) 75
 PC 4 SETH 90 ST
 ANIMATE.BALLOON.2
END

TO ANIMATE.BALLOON.2
 REPEAT 8 [FD 3 RT 90 FD 1 LT 90]
 REPEAT 4 [FD 1]
 REPEAT 8 [FD 3 LT 90 FD 1 RT 90]
 REPEAT 4 [FD 1]
 ANIMATE.BALLOON.2
END

TO FLY.BALLOON.3
 DRAW ERASE.SPRITES HT
 FULLSCREEN BG 15
 TELL 7
 SETXY (−120) 75
 PC 4 SETH 45 ST
 ANIMATE.BALLOON.3
END

TO ANIMATE.BALLOON.3
 REPEAT 45 [FD 1 RT 2]
 REPEAT 45 [FD 1 LT 2]
 ANIMATE.BALLOON.3
END
```

Let's try an animation project with an airplane. Shape 5 in the VEHICLES file is an airplane pointing left. Use it in a set of procedures that make the airplane move across the top of the screen from right to left. Then, modify these procedures so that the airplane appears to take off from ground level at a height of −75 on the Y coordinate. Make the airplane rise slowly as it moves across the screen and level off at a constant height when its Y coordinate reaches 75.

One way to make the airplane level off is to use the following two statements within your animation subprocedure:

```
FD 1
IF YCOR < 75 THEN LT 90 FD 1 RT 90
```

As long as the sprite is below a screen height of 75 on the Y coordinate, the condition in the second statement will be satisfied and the sprite will alternately move both forward and upward. Once it reaches a height of 75, however, the condition is no longer satisfied and only the forward motion will continue.

Once you're able to make the airplane rise and level off, add another subprocedure or two that use a hidden turtle to draw a runway and an airport building on the screen before the animation begins. If you want, you can save the background display as a picture on your Work Disk and have your main procedure read it in the way I did with SEA.SCENE.2.

These are just a few of the animation effects you should now be able to achieve using what you've learned about Commodore 64's sprite capabilities. Think up some animation projects of your own and use any of the shape files on the Utilities Disk in them. Then, save the background pictures you've developed on your Work Disk using the SAVEPICT command and the procedures you've developed on your Work Disk using the SAVE command, so they won't be lost when you type GOODBYE or turn off the computer. That way, you can read them back into your workspace to either run them or modify them any time you want.

Logo Activity Time

The following activity, called RELAY, is a self-starting relay race between four sprites that uses the vehicle shapes introduced in this chapter. It also uses a file on the Utilities Disk called SPRITES. You'll learn more about the SPRITES file in Chapter 12, but for now, we'll just use it. So, before you run RELAY, be sure you've loaded both the SPRITES file and the VEHICLES file into your workspace from the Logo Utilities Disk.

RELAY begins with Sprites 1, 2, 3, and 4 assigned Shape 7 (a balloon) and lined up one below the other at the left edge of a purple rectangle. Each sprite has been given a different size and a different color, with Sprite 1 wide, tall, and blue; Sprite 2 wide, short, and red; Sprite 3 narrow, tall, and green; and Sprite 4 narrow, short, and yellow, as shown in Figure 11-5 (of course, the photo doesn't show the colors, but it does give you an idea of the size and shape).

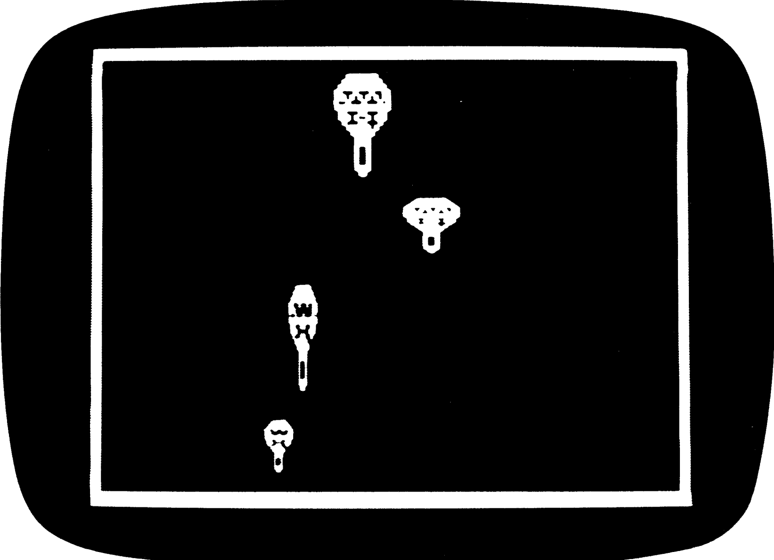


Figure 11-5. The screen at the start of RELAY

Once the four sprites are positioned, they begin moving across the screen from left to right at different speeds. As each sprite reaches the right border of the rectangle, it changes shape and reverses direction. The number of each shape is automatically reduced by one. The same thing happens to the sprites every time they reach the left or right border of the rectangle. By changing the shape of each sprite from Shape 7 all the way down to Shape 1, the procedure gives the impression of a relay race with all seven vehicle shapes appearing in reverse order.

The first sprite to become a truck (Shape 1) and reach the finish line is the winner. When that happens, a subprocedure automatically returns the sprites to their standard sizes and shapes, and the screen to its standard format.

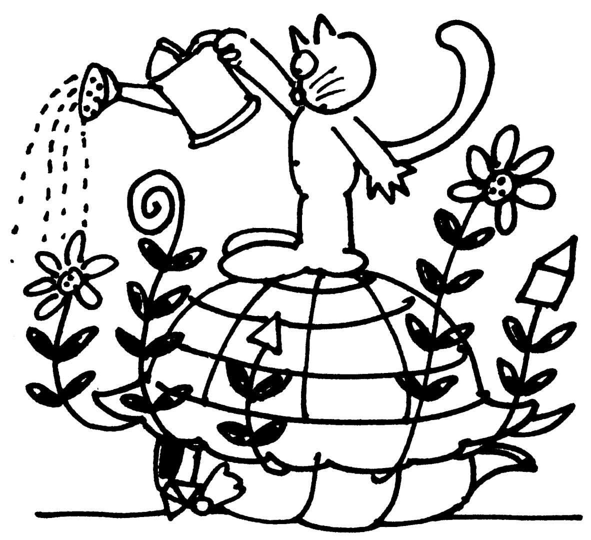
To use RELAY, insert your Activity Disk into your disk drive and type READ "RELAY on your keyboard. The file is self-starting, so, once all of its procedures are read in, it begins immediately without your having to do anything further. Once it's been read in the first time, of course, you can run it over and over by typing the name of the main procedure, RELAY. You might find it interesting, after seeing how RELAY looks with vehicle shapes, to try it with one or more of the other shape files. The animal shapes, in particular, give a striking effect. You can even design and save a special file of shapes created expressly for use with RELAY.

To make the races more realistic and unpredictable, the animation subprocedure has been designed to randomly select a number between 0 and 20 as the distance for the computer to use each time a sprite is moved. It does this using a built-in, primitive Logo command called RANDOM. You'll see, how RANDOM works and how to use it in a later chapter.

New Commands Introduced In This Chapter

Command	*Abbreviation*
TELL	
SETSHAPE	
WHO	
SHAPE	
MAKE "STARTUP	

Growing and Cultivating Sprites



As you learned in the last chapter, Logo offers quite a variety of sprite shapes in the sprite files on the Utilities Disk. In this chapter, you'll learn how to modify the sprite shapes by stretching them horizontally and vertically. You will also learn how to create new sprite shapes of your own design, using procedures that are included on the Utilities Disk.

In addition, you'll learn how to tell when a sprite is touching another sprite or part of a drawing. The ability to detect such sprite "collisions" is helpful in creating games in which the object is to avoid running into things.

Stretching Sprites

Let's begin by learning how to enlarge a sprite both horizontally and vertically, using a procedure in a file on the Utilities Disk called SPRITES.

To use the SPRITES file, put the Utilities Disk in your disk drive, close the door, and type READ "SPRITES.

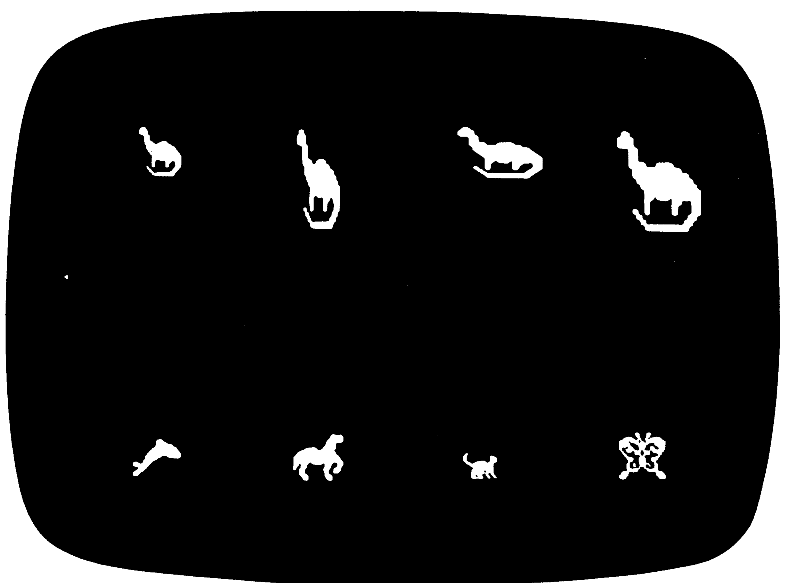
When you first load a set of sprite shapes from the Utilities Disk, all of the shapes are the smallest they can be. You can make a sprite bigger by using the TELL command to make the sprite with the shape you want become the current sprite, and then using the BIGX and BIGY procedures provided in the SPRITES file. As you'll see, BIGX doubles the size of the current sprite horizontally (in the X direction) by expanding the shape to the right. BIGY doubles the size of the current sprite vertically (in the Y direction) by expanding down. Once a sprite's shape has been doubled in either or both of these directions, you can reduce it back to its original size using the SMALLX and SMALLY procedures, which are also in the SPRITES file. SMALLX reduces a sprite by half horizontally (in the X direction) and SMALLY reduces it by half vertically (in the Y direction). Using these size procedures, each shape can have four possible sizes:

- Narrow and short (SMALLX and SMALLY)
- Narrow and tall (SMALLX and BIGY)
- Wide and short (BIGX and SMALLY)
- Wide and tall (BIGX and BIGY)

To see for yourself what these size procedures can do, transfer the animal shapes into your workspace by typing

READ "ANIMALS and use the procedure DISPLAY.SPRITES to show all the shapes on the screen at the same time. If DISPLAY.SPRITES isn't in your workspace, you will need to load it from your Work Disk. Now, use TELL and SETSHAPE with Sprites 0, 2, and 3 to assign each one the same shape as Sprite 1, the dinosaur. Finally, use the SMALLX, SMALLY, BIGX, and BIGY procedures to make Sprite 0 narrow and short, Sprite 1 narrow and tall, Sprite 2 wide and short, and Sprite 3 wide and tall. Your display screen should look like the illustration in Figure 12-1.

Figure 12-1. The dinosaur shape in four sizes



Turtle Trap: Once the size of a sprite's shape has been changed, it remains that size, even if you assign it a new shape or you load a new shape file from the Utilities Disk. If you make Sprite 1 a tall, thin dinosaur and then read in the VEHICLES file, Sprite 1 will be a tall, thin truck. If you want the truck to be its original short, thin size, you have to make the change using the command SMALLY.

The Utilities Disk contains three demonstration files that use these size commands and illustrate the kinds of animated projects you can develop with them. The names of these files are DINOSAURS, RUNNER, and SUBMARINE. DINOSAURS shows a family of three dinosaurs on a walk with three trees in the background. RUNNER shows a woman jogging. SUBMARINE, reminiscent of the animated Beatles film *Yellow Submarine,* shows a submarine changing color and size as it travels through an ocean that also changes colors. These files are self-starting, so, when you read them in, they will begin to run automatically. Take a quick look at DINOSAURS, RUNNER, and SUBMARINE, then we'll learn how to modify sprite shapes.

Editing Sprite Shapes

In addition to providing the predefined sprite shapes used up to now, Commodore 64 Logo also allows you to modify, or edit, these predefined shapes and even create completely new shapes of your own design. Editing sprite shapes requires use of a procedure called EDSH (short for EDit SHape). You can find this procedure in a file called SPRED (short for SPRite EDitor) on the Utilities Disk.

SPRED contains the same procedures as SPRITES. So, by reading in SPRED, you automatically get all of SPRITES, as well as the capability of editing and creating shapes.

Let's use the vehicle shapes to see what EDSH does. Make sure the vehicle shapes are in your workspace, then type READ "SPRED.

Editing Predefined Shapes

To modify a predefined shape, make the sprite with that shape the current sprite and give it a color that contrasts with the background of the Sprite Editor mode. Then, enter the Sprite Editor mode by typing EDSH. For example, suppose

Turtle Trap: As you'll see in the next section, when you enter the Sprite Editor mode to edit predefined shapes, your screen shows both a large-sized version of the shape composed of dots in a grid on the left of the screen and a normal-sized version of the shape to the right of the grid. Unfortunately, the background color of the Sprite Editor mode will probably be light blue (color number 14), the same color as the sprite shapes when they're first read into the computer from their shape files. So, when you enter the Sprite Editor mode, the normal-sized version of the shape blends in with the background and you can't see it.

To make the normal-sized shape visible in the Sprite Editor mode, change the shape's color to white (or any other color that shows up clearly on a light blue background) by typing PENCOLOR 1 (or PC 1) before entering the Sprite Editor mode. Or, since the Sprite Editor uses the text screen, just use TEXTCOLOR and TEXTBG to make the text and its background any colors you want.

you want to edit the truck shape. You'd put it in the Sprite Editor mode and make it white, so you can modify it. Since the truck is Shape 1 and Shape 1 currently is assigned to Sprite 1, you can put the truck in the Sprite Editor mode by typing TELL 1 PC 1 EDSH. Your display looks as in Figure 12-2,

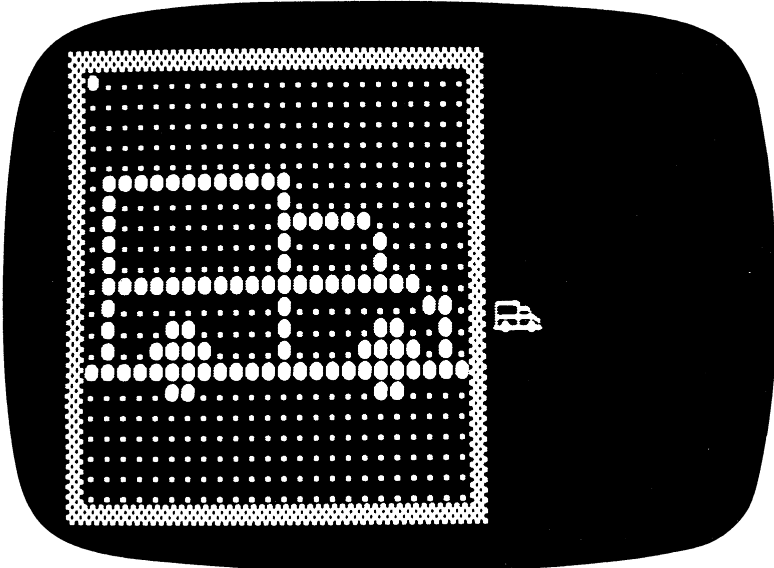


Figure 12-2. The truck shape in the Sprite Editor

with the normal-size truck shape appearing in white just to the right of center, and an enlarged version of the truck drawn in dots inside a Sprite Editor box filling most of the left side of the screen.

The Sprite Editor box consists of a grid 24 dots wide and 21 dots high. Each dot in this grid represents one of the tiny rectangles, called pixels, that make up the display screen. Recall that pixels were mentioned in Chapter 6 in the discussion of screen pictures. If you look closely at the screen, you'll notice these pixels. As you can see, the pixels are small and numerous. So, the area of the screen represented by the 24 by 21 dot grid is only a small part of the screen. The smaller dots in the Sprite Editor box represent pixels that will be filled in with the background color, while the larger dots represent pixels that will be filled in with the pen color to make the truck shape on the screen. (For simplicity, I'll refer to the small dots as "empty" pixels and the large dots as "filled-in" pixels, even though the smaller dots aren't, in fact, empty.) If you compare the grid inside the Sprite Editor box with the truck shape to its right, you'll notice that the truck shape to the right is a smaller version, but otherwise an exact duplicate of the shape formed by the colored-in pixels in the grid.

The large blinking dot in the upper left corner of the Sprite Editor box is the Sprite Editor cursor. You move the Sprite Editor cursor around the grid using the <CRSR> keys the same way you move the rectangular cursor when you type commands and write and edit procedures. The <HOME> key can be used to immediately send the cursor back to its initial position at the upper left corner of the grid. The <RETURN> key can be used to immediately send the cursor to the first pixel on the next line.

You can modify the truck shape (or any other sprite shape) by using the Sprite Editor cursor to fill in any empty pixels in the grid with color and erase the color from any of the pixels in the grid that are already filled in. Any changes

you make in the pixels of the Sprite Editor box are immediately reflected in the corresponding normal-size shape to the right of the grid. So, you can see the results of your editing as you're doing it.

To fill in a pixel, use the < * > or < + > key. Pressing < * > fills in the pixel the cursor is on and moves the cursor to the right one space. Pressing < + > fills in the pixel the cursor is on without moving the cursor.

To empty a pixel, use the space bar, the < – > key, or the <DEL> key. Pressing the space bar empties the pixel the cursor is on and moves the cursor one space to the right. Pressing < – > empties the pixel the cursor is on without moving the cursor. Pressing <DEL> empties the pixel to the left of the cursor and moves the cursor into that position.

In addition to these keys, several other keys are useful in the Sprite Editing mode. For example, you might want to see what your modified sprite shape would look like if you later decide to change its dimensions using the commands SMALLX, SMALLY, BIGX, and BIGY. You can do this by pressing the <X> key to enlarge the shape horizontally (in the X direction), and by pressing the <Y> key to enlarge the shape vertically (in the Y direction). Press the same key you used to enlarge a shape to reduce the shape to its original smaller size. Compare the screen displays in Figures 12-2, 12-3, and 12-4. Figure 12-2 shows what the screen looks like

when you first enter the Sprite Editor mode. Figure 12-3 shows what the screen looks like after pressing <X>, with the truck in its BIGX size, and Figure 12-4 shows what the screen looks like after pressing <Y>, with the truck in its BIGY size.

Figure 12-3. The truck shape in BIGX size

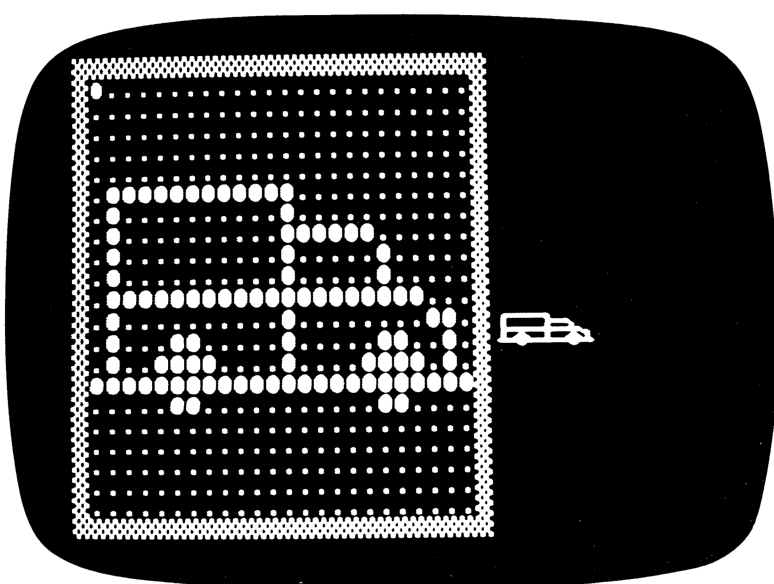
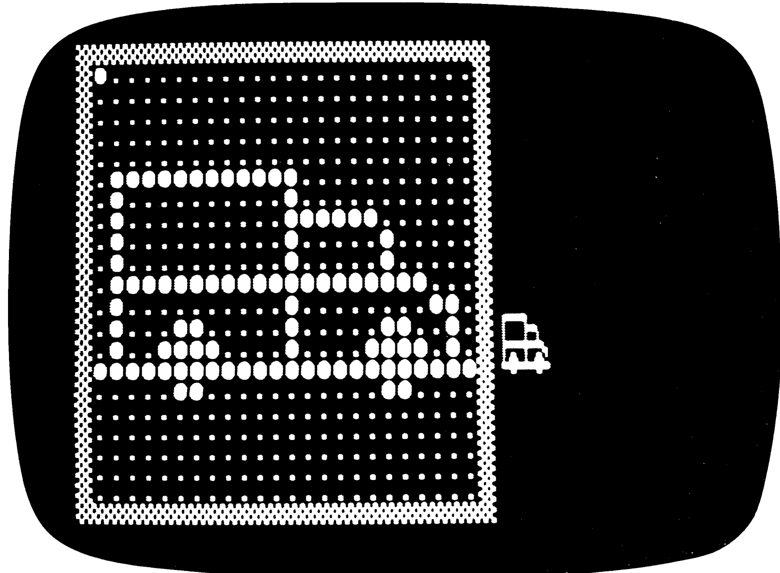
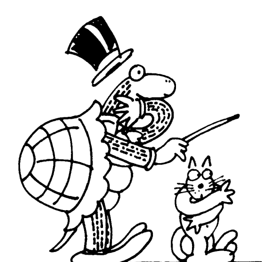


Figure 12-4. The truck shape in BIGY size





Turtle Trap: In an earlier chapter, you learned that, when editing procedures, you could exit the Edit mode without implementing any of your changes by pressing <CTRL> G. That's not the case when you edit sprite shapes. Once you've changed a sprite shape in the Sprite Editor mode, the only way to undo the change is to use the Sprite Editor cursor to restore the original shape one pixel at a time or to read the original set of shapes back into your workspace from the Utilities Disk.

Another sprite-editing command that's sometimes useful is <CTRL> 9, which reverses the pixels in the shape—that is, all filled-in pixels are emptied and all empty pixels are filled in. Try using <CTRL> 9 now to see the effect it has on the truck shape. To return the truck to its starting shape, simply press <CTRL> 9 again.

Once you've modified the shape to look the way you want it to look, you can exit the Sprite Editor mode and implement the change by pressing <CTRL> C or the <RUN/STOP> key on the far left of the keyboard in the second row from the bottom. After you implement the changes, any sprite that you assign the number of the shape you placed in the Sprite Editor box will have the modified shape. To restore the original shape, you must either modify it back to that shape using the Sprite Editor mode or reload the original shapes from the Utilities Disk.

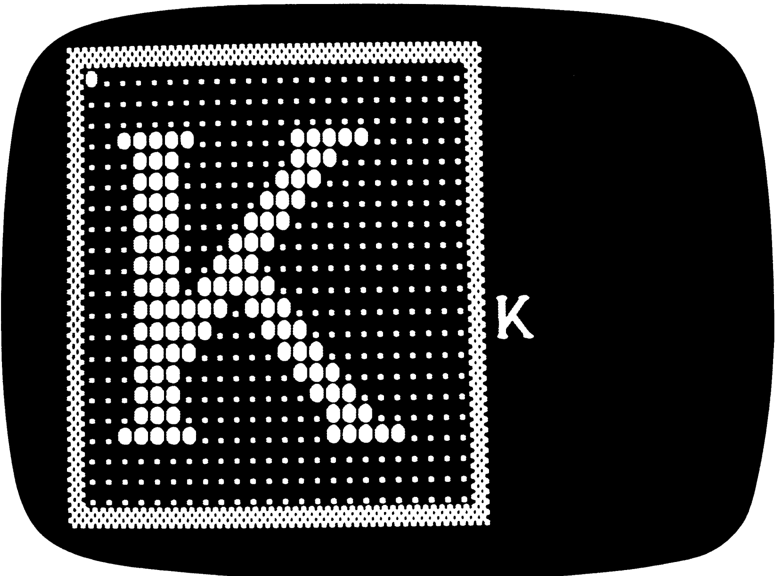
Creating Sprite Shapes

If you want, you can create your own shapes, rather than simply modifying the predefined shapes. But you need to start with one of the predefined shapes so that your shape has a number.

First, put the sprite with the shape you want to change into the Sprite Editor mode. For example, if you want to create a new Shape 1, put Sprite 1 with Shape 1 (the truck, if the VEHICLES file is the one in your workspace) into the Sprite Editor mode, just as if you wanted to modify it. Once Shape 1 is in the Sprite Editor mode, empty all the pixels in it, leaving

a blank shape, by using the <CLR> key. The <CLR> key is the same key as the <HOME> key. But it does a different job when you use it with the <SHIFT> key. You can make it do its <CLR> key job—emptying all of the pixels in the Sprite Editor box—by holding down the <SHIFT> key and pressing the <CLR> key at the same time. You can now fill in those pixels that will give you the shape you want Shape 1 to have. The normal-sized shape appears to the right of the grid as the pixels are filled. For example, in Figure 12-5, I cleared all of the pixels from Shape 1 by pressing the <SHIFT> and <CLR> keys. Then, I used the Sprite Editor cursor to fill in pixels to form the first letter of my first name, K. Then, I pressed <CTRL> C (I could have pressed the <RUN/STOP> key) to exit the Sprite Editor, making Shape 1 the letter K that I had created. From that point on, any sprite that I assigned Shape 1 looked like this K. To restore the original truck shape to Shape 1, I had to load the VEHICLES file, containing the truck as Shape 1, into my workspace. I could also have put the K back into the Sprite Editor mode and reconstructed the truck shape. But simply loading the VEHICLES file again was easier.

Figure 12-5. A "K" shape in the Sprite Editor



The sprite-editing keys and commands discussed in this chapter are listed in Figure 12-6 for ease of reference.

Figure 12-6. Sprite-editing keys and commands

Command	*Purpose*
<CRSR⇕>	Moves the sprite cursor up and down without changing any of the pixels.
<CRSR⇄>	Moves the sprite cursor left and right without changing any of the pixels.
<HOME>	Moves the sprite cursor to its initial position at the upper left corner of the Sprite Editor box.
<RETURN>	Moves the sprite cursor to the leftmost pixel on the next line.
< * >	Fills the pixel the sprite cursor is on and moves the cursor one pixel to the right.
<+>	Fills the pixel the sprite cursor is on without moving the cursor.
<SPACE BAR>	Empties the pixel the sprite cursor is on and moves the cursor one pixel to the right.
<–>	Empties the pixel the sprite cursor is on without moving the cursor.
<DEL>	Empties the pixel to the left of the sprite cursor and moves the cursor to that position.
<X>	Switches the shape from SMALLX to BIGX size. Press a second time to restore to SMALLX size.
<Y>	Switches the shape from SMALLY to BIGY size. Press a second time to restore to SMALLY size.

(continued)

Figure 12-6. Sprite-editing keys and commands (continued)

Command	*Purpose*
<CLR>	Empties all the pixels in the Sprite Editor box.
<CTRL> 9	Reverses all the pixels in the Sprite Editor box. Press it a second time to restore the pixels to their original state.
<CTRL> C	Exits the Sprite Editor mode and implements any shape changes made.
<RUN/STOP>	Same as <CTRL> C.

Saving Sprite Shapes

Once you've modified predefined sprite shapes or created new shapes of your own, you can save the current set of seven shapes on your Work Disk as a shape file. Type the command SAVESHAPES, a space, a quotation mark, and the name you want to give to your shape file. For example, suppose you've left Shapes 3 through 7 from the VEHICLES file as they were read in from the Utilities Disk, but you've modified Shape 1 and created a completely new shape for Shape 2. You can save this set of shapes under the file name MINE by inserting your Work Disk into the disk drive and typing:

```
SAVESHAPES "MINE
```

You can then read this shape file back into your workspace from your Work Disk at any future time by typing:

```
READSHAPES "MINE
```

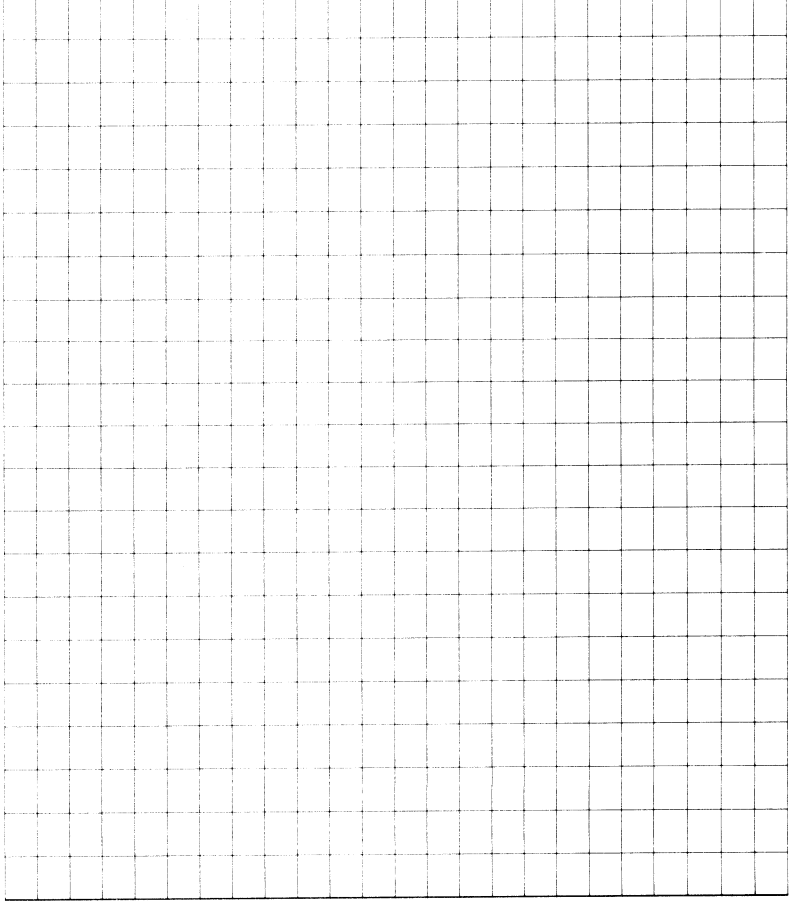
You should know that the READSHAPES and SAVESHAPES commands from the Sprite Editor neither save nor recall the names of the shapes in the predefined shape

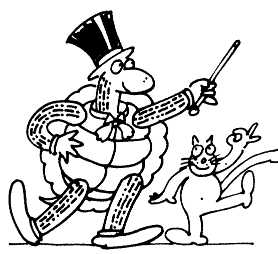
files on the Utilities Disk (SHAPES, ANIMALS, VEHICLES, or ASSORTED). These names are created by procedures on the Utilities Disk that define the names for the shapes and then load them. If you're curious, you could, for example, type GOODBYE and READ "ANIMALS, and then type PO ALL to see exactly how this is done.

More Things to Do

As mentioned earlier in this chapter, using graph paper to design new sprite shapes is simpler and more efficient than editing on the screen. For your convenience, if you don't have any graph paper handy, Figure 12-7 is a graph of the same

Figure 12-7. A graph for designing sprite shapes





Turtle Tip: Creating new sprite shapes can be a lot of fun, especially if you're creating animated figures for use in computer games of your own design. You can save these new shapes in a shape file and read them back into your workspace whenever you want to play the game. (You'll learn how to save shape files on a disk and read them back into your workspace in the next section of this chapter.)

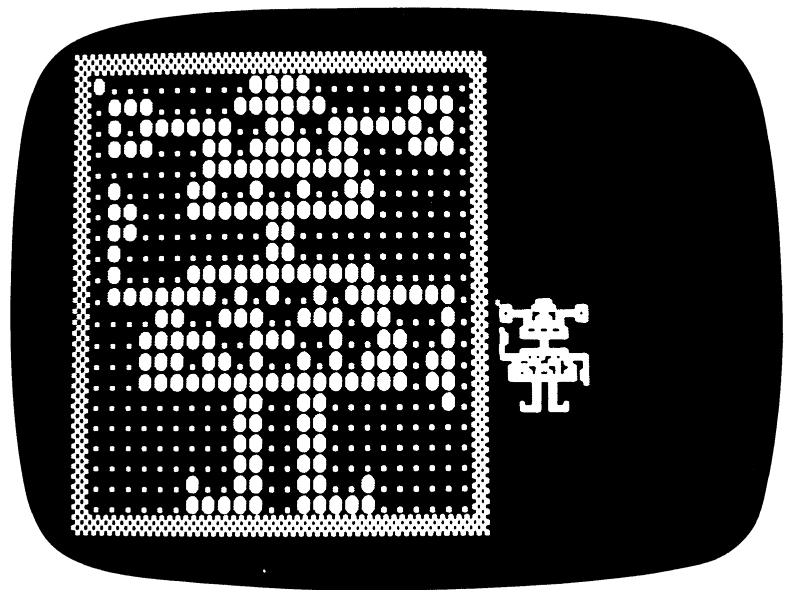
For example, my five-year-old son Timothy is fascinated with toy robots right now. So, I asked him if he'd like to create a Logo game that would have a robot as the main character. Of course, he was thrilled with the idea. So, one afternoon, we loaded Logo into the computer and got to work creating a sprite shape that would resemble his favorite toy robot.

We began by drawing a picture of the Sprite Editor grid on a piece of paper to use in planning the robot's shape. We then used a pencil to fill in pixels on the grid until the shape in the grid looked the way we wanted the robot to appear on the display screen. We used a pencil instead of a pen or a crayon to fill in the pixels so that we could erase filled-in pixels that didn't look right to us.

When we were satisfied with the shape we had developed, we followed the same procedure that was described earlier in this chapter for creating the letter K sprite shape. We then filled in the pixels of the Sprite Editor box to match the picture we had created on the paper grid and pressed <CTRL> C to exit the Sprite Editor mode and implement the new shape. A picture of the Sprite Editor box with the pixels filled in to create the robot shape is shown in Figure 12-8.

Once we had the robot shape, I developed a main procedure that drew the picture of a robot fortress on the screen, loaded the shape file with the robot shape in it into the workspace, assigned the robot shape to Sprite 0, and displayed the robot at the center of the screen. I then created several simple procedures to allow Timothy to control the robot with single-key commands. For example, he could move the robot to different rooms of the fortress by pressing the key with the first letter of the room's name on it; make the robot big or small by pressing the <B> or <S> keys; tell the robot to leave the fortress by pressing the <G> key for "Goodbye"; or tell it to return to the fortress by pressing the <H> key for "Hello."

Figure 12-8. A robot shape in the Sprite Editor box



If you want to, you can now use the photo in Figure 12-8 as a guide in creating the robot shape on your own computer. Then, after you've learned how to save a shape file on a disk in the next section, you can save the robot shape with the other shapes in your workspace on your Work Disk. Or, you can plan and create other sprite shapes of your own design. If your children have certain favorite toys they like to play with, you can help them bring these playthings to life by creating sprite shapes in their images and using those shapes in Logo games you create together.

dimensions as the Sprite Editor box (24 pixels wide by 21 pixels high) that you can use for your designing.

If you're going to edit a predefined shape, start by putting the shape in the Sprite Editor mode using the EDSH procedure. Then, use a pencil to fill in all those boxes on the graph in Figure 12-7 that correspond to filled pixels in the Sprite Editor box. Now, you can plan your shape modifications by using the pencil to fill in empty boxes, and an eraser

to empty filled-in boxes, until the picture on the graph is the shape you want. It's then a simple process to use the shape-editing commands to fill and empty pixels in the Sprite Editor box to match the filled and empty boxes on the graph. Similarly, to create a completely new sprite shape, as we did with the robot shape, just start with the graph empty. Then, enter the shape you want to use into the Sprite Editor mode, press <CLR> to empty all the pixels in the Sprite Editor box, and use the shape-editing commands to fill in those pixels that correspond to the filled-in boxes on the graph.

By using graph paper to plan your shapes, you can concentrate on the shapes themselves without having to worry about the correct commands for moving the Sprite Editor cursor around the Sprite Editor box, filling and emptying pixels.

After you've used this graph to modify and create sprite shapes, try saving these shapes on your Work Disk with the SAVESHAPES command. Read them back into your workspace using the READSHAPES command. Make sure you can do this correctly so that later on, when you spend a lot of time and effort creating special sprite shapes you really want to keep, you won't lose them because you never quite learned how to use the SAVESHAPES and READSHAPES commands.

Detecting Sprite Collisions

As you saw in Chapter 11, when a sprite passes a part of the screen that has a drawing on it, the sprite passes in front of the drawing without affecting it. Similarly, when two sprites meet, the lower-numbered sprite appears to pass in front of the higher-numbered sprite. But there are times, particularly in designing games, when it would be useful to be able to detect such "collisions" and have the computer take appropriate actions when they occur.

For example, suppose you've drawn a racetrack on the screen and you're using two or three of the sprites as race cars. It might be fun to make the screen change colors several times and display the word CRASH every time one of the race cars comes in contact with the border of the track or another race car. The SPRITES file on the Utilities Disk provides two procedures that can be used to detect these kinds of collisions: TB? (short for Touching Border?) and TS? (short for Touching Sprite?).

TB? and TS? are conditional procedures. TB? asks the computer to check whether the current sprite is touching a part of the screen that's colored differently from the background color. If the sprite is touching a different-colored part of the screen, TB? gives the result TRUE. Otherwise, TB? gives the result FALSE. Similarly, TS? asks the computer to check whether the current sprite is touching another sprite. If it is, TS? gives the result TRUE. Otherwise, TS? gives the result FALSE.

The following procedures illustrate the use of these collision-detection commands. The main procedure HIT.WALL, and its subprocedures DRIVE.CAR and BOOM, set the screen to show a wall on the right and a car moving toward the wall from the left. A conditional statement containing TB? is used to check whether the car has collided with the wall. As long as the answer is "No," the car keeps moving. But when the answer becomes "Yes," the car bounces back a short distance and the procedures stop.

The main procedure CRASH, and its subprocedures DRIVE.CAR.TRUCK and CRUNCH, set the screen with a car in front of a truck, both moving from left to right. The truck moves faster than the car, so eventually the two collide. When they collide, a conditional statement containing TS? causes the screen to flash different colors and the procedure to end.

Both HIT.WALL and CRASH use ERASE.SPRITES as a subprocedure, so the procedure listing for ERASE.SPRITES is given here, as well. Enter HIT.WALL, CRASH, and their subprocedures into your computer (and ERASE.SPRITES, as well, if it's not already in your workspace). Then, one at a time, run both main procedures by typing first HIT.WALL and then CRASH. These procedures are intended to be used with the VEHICLES shape file, but you can try them with another shape file on the Utilities Disk (or a shape file of your own design) if you want to see what such a change looks like. The initial screen displays for these procedures, if vehicle shapes are used, are shown in Figures 12-9 and 12-10.

```
TO HIT.WALL
 DRAW ERASE.SPRITES HT
 CLEARTEXT BG 9
 TELL 2 PC 5 BIGX
 SETXY (-120) 0 SETH 90 ST
 TELL 0 PC 8 PU
 SETXY 100 (-15) SETH 0 PD
 REPEAT 10 [FD 65 BK 65 RT 90 FD 1 LT 90]
 PC 1 PU HOME
 TELL 2
 DRIVE.CAR
END

TO DRIVE.CAR
 IF TB? THEN BOOM
 FD 4
 DRIVE.CAR
END

TO BOOM
 BACK 50
 SMALLX
 TOPLEVEL
END
```

```
TO CRASH
 DRAW ERASE.SPRITES HT
 CLEARTEXT BG 9
 TELL 1 PC 15 BIGX
 SETXY (-120) 0 SETH 90 ST
 TELL 2 PC 5 BIGX
 SETXY 0 0 SETH 90 ST
 DRIVE.CAR.TRUCK
END

TO DRIVE.CAR.TRUCK
 IF TS? THEN CRUNCH
 TELL 2 FD 2
 TELL 1 FD 5
 DRIVE.CAR.TRUCK
END

TO CRUNCH
 FULLSCREEN
 TELL 1 HT
 TELL 2 HT
 REPEAT 5 [REPEAT 5 [BG 2] REPEAT 5 [BG 1]
  REPEAT 5 [BG 6]]
 ERASE.SPRITES
 TOPLEVEL
END

TO ERASE.SPRITES
 MAKE "NUMBER 0
 REPEAT 8 [TELL :NUMBER PU HT HOME MAKE
  "NUMBER :NUMBER+1]
 TELL 0 ST PD SPLITSCREEN BG 11
END
```

Figure 12-9. The screen at the start of HIT.WALL

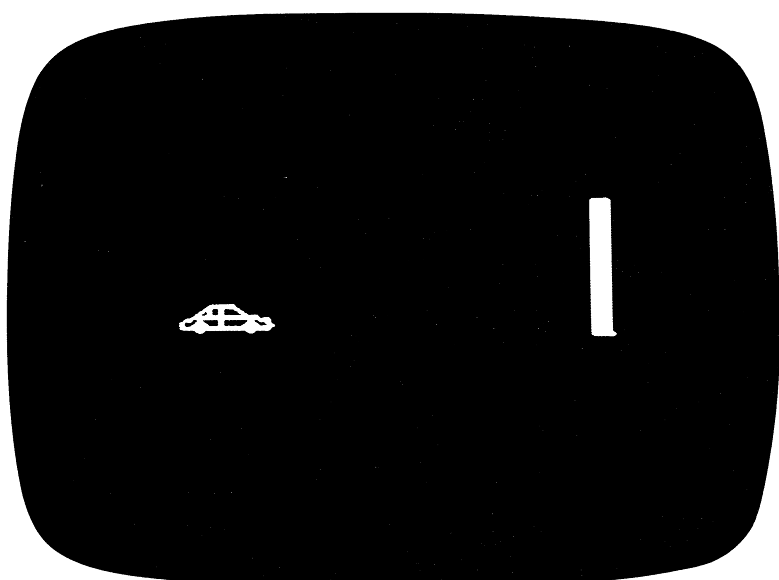
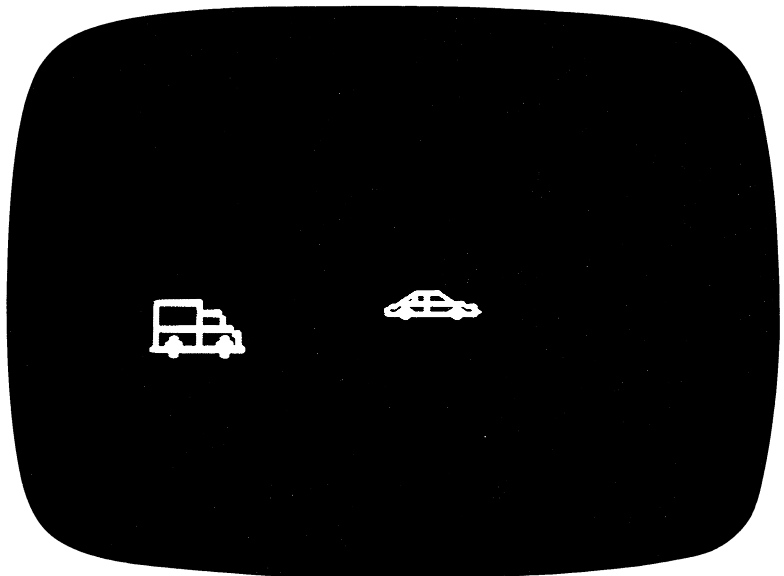


Figure 12-10. The screen at the start of CRASH



Remember, the collision-detection commands introduced here aren't built-in Logo primitives; they're procedures provided in the file called SPRITES on the Logo Utilities Disk. If you want to use them, you must either read the SPRITES file

into your workspace from the Utilities Disk or save them on your Work Disk as a part of your own procedure file and read them into your workspace from your Work Disk.

Logo Activity Time

The activity BUILDBLOCKS on your Activity Disk is based upon a pre-school and elementary school activity in which children are given a number of geometric shapes cut out of colored paper. The children are encouraged to use these shapes to form interesting and colorful designs. Similarly, BUILDBLOCKS uses a sprite shapes file, also called BUILDBLOCKS, to assign a variety of geometric shapes to the eight Commodore 64 Logo sprites. The activity allows you to change the sprites' colors, hide them, show them, and move them around the screen using special shorthand procedures that are part of the BUILDBLOCKS file.

Since BUILDBLOCKS uses several special procedures from the SPRED file on the Logo Utilities Disk, the SPRED file must be loaded first. To do this, simply insert the Utilities Disk in your disk drive, close the disk drive door, and type:

```
READ "SPRED
```

The red light on the disk drive goes on to indicate this file is being read into your workspace. When the red light goes off, you can take the Utilities Disk out of the disk drive and replace it with your Logo Activity Disk. Now, type:

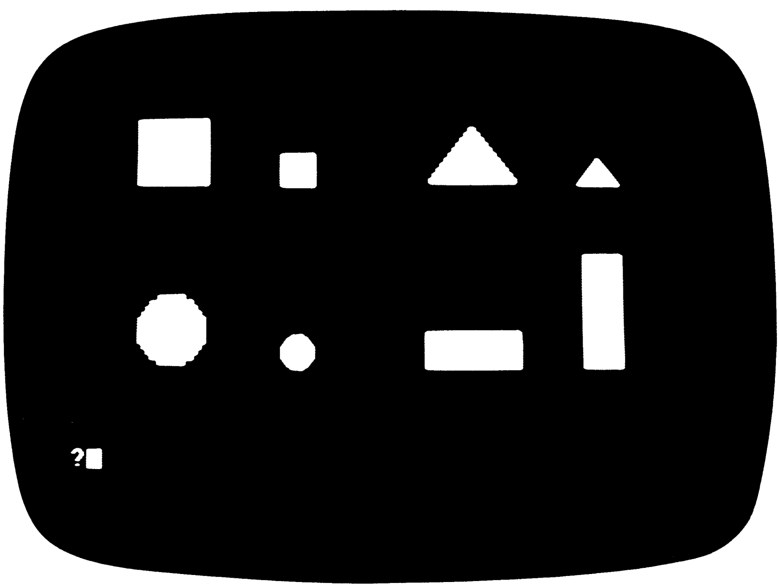
```
READ "BUILDBLOCKS
```

on your keyboard and this activity will be read into your workspace as well.

When the red light on the disk drive goes off and the cursor reappears on your screen, you can run the main procedure in the activity, called BUILDBLOCKS, by typing BUILDBLOCKS and pressing <RETURN>. Make sure the Activity Disk is still in the disk drive when you do this because the BUILDBLOCKS procedure automatically reads

the geometric shapes file into your workspace as soon as you run it. Once the shapes file is read into your workspace and assigned by the procedure to the appropriate sprites, the eight shapes will be displayed on your screen as illustrated in Figure 12-11. The eight figures consist of a big square, a small square, a big triangle, a small triangle, a big circle, a small circle, a horizontal rectangle, and a vertical rectangle. All are white on a black background.

Figure 12-11. The screen at the start of BUILDBLOCKS



You can now use the shorthand, three-letter commands that are built into the activity as procedures to change the figures' colors, to hide them, to show them again, or to move them up, down, left, and right. The first two letters of each command indicate the particular figure to which the command is addressed. The third letter of each command indicates the action that figure is to take. For example, to change the color of the big square to blue you would use the command BSC 6 (BS for big square, C for change color, and 6 for blue, since blue is color number 6 in Logo). Similarly, to hide the small triangle you would use the command STH (ST for small triangle and H for hide). To move the vertical rectangle

25 steps to the right you would use the command VRR 25 (VR for vertical rectangle, R for move to the right, and 25 for the number of steps the figure is to move). These two-letter names and one-letter actions are listed in Figure 12-12. A list of Commodore 64 Logo's colors and color numbers is also provided for ease of reference.

Figure 12-12. Commands and colors used in BUILDBLOCKS

Names

Command	*Sprite*
BS	Big Square
SS	Small Square
BT	Big Triangle
ST	Small Triangle
BC	Big Circle
SC	Small Circle
HR	Horizontal Rectangle
VR	Vertical Rectangle

Actions

Command	*What It Does*
C	Changes the figure's color to the specified color.
H	Hides the figure from view.
S	Shows the figure.
U	Moves the figure up the specified number of steps.
D	Moves the figure down the specified number of steps.
R	Moves the figure to the right the specified number of steps.
L	Moves the figure to the left the specified number of steps.

(continued)

Figure 12-12. Commands and colors used in BUILDBLOCKS (continued)

Commodore 64 Logo Colors

Color	*Number*
Black	0
White	1
Dark Red	2
Cyan (greenish-blue)	3
Purple	4
Dark Green	5
Dark Blue	6
Yellow	7
Orange	8
Brown	9
Light Red	10
Dark Gray	11
Medium Gray	12
Light Green	13
Light Blue	14
Light Gray	15

Use these commands to select the colors you want to use, hide the geometric figures you don't want to use, and move the remaining geometric figures around the screen to create interesting patterns and designs. Two examples of the results you can obtain in this activity are shown in Figures 12-13 and 12-14.

When you've finished using this activity, type ERASE.SPRITES. This procedure, which is contained in the BUILDBLOCKS file, moves all the sprites back to the Home

position at the center of the screen, hides all the sprites except Sprite 0, and gives Sprite 0 back its original turtle shape.

Figure 12-13. A design created by BUILDBLOCKS

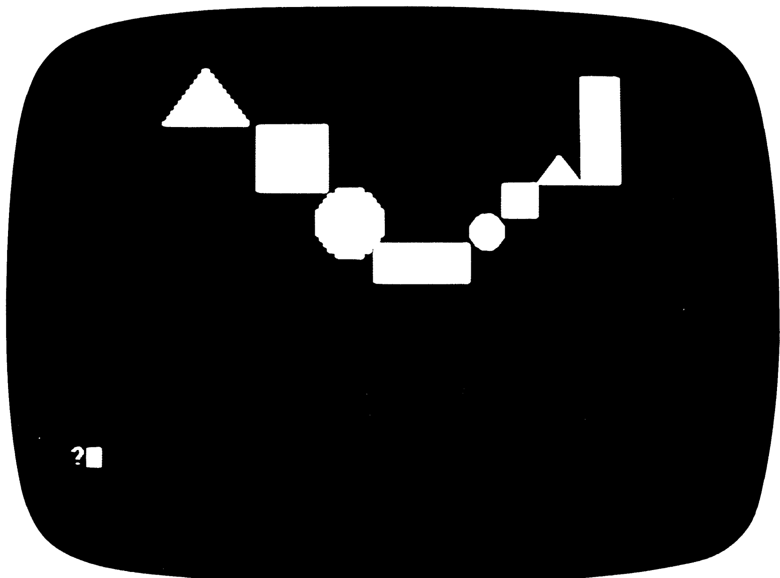
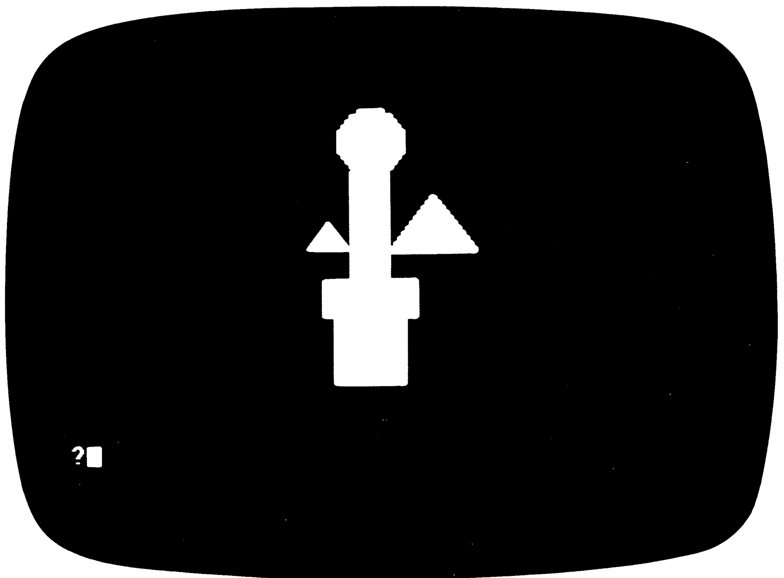


Figure 12-14. Another design created by BUILDBLOCKS



New Commands Introduced In This Chapter

There were no new commands introduced in this chapter, but you were introduced to the following helpful procedures:

In the SPRED file on the Utilities Disk: SAVESHAPES, READSHAPES, SMALLX, SMALLY, BIGX, BIGY, EDSH, TB?, and TS?.

All of the above procedures, except SAVESHAPES and EDSH, are also contained in the SPRITES file on the Utilities Disk.

Working With Numbers

If asked what computers do, most people probably would reply, "Compute." After all, that's the root word of "computer." As you've seen so far, Logo allows you to do more with your Commodore 64 computer than just add, subtract, multiply, and divide. But it doesn't skimp on math capabilities. It performs all of the standard mathematical operations. Let's look at the basic ones.

Arithmetic Operations

You can perform the four standard arithmetic operations with Logo by simply typing in a problem, or expression, and pressing <RETURN>. The keys used for these operations are:

- < * > for multiplication
- < / > for division
- <+> for addition
- <−> for subtraction

Recall from earlier chapters that the symbol for multiplication in Logo is * rather than an uppercase or lowercase X, and the symbol for division is / rather than the ÷ symbol.

Since the four arithmetic operations are taught in elementary school, the material in this section should be appropriate for an elementary-school-age child. Generally, addition and subtraction are introduced in the first through third grades (ages 6 through 8) and multiplication and division in the fourth through sixth grades (ages 9 through 11).

Let's use Logo to do a few computations. Type in a mathematical expression and press <RETURN>. The computer will evaluate the expression and print RESULT:, a space, and the result. For example, try typing:

```
1+2+3
```

The screen displays:

```
RESULT: 6
```

You can explicitly instruct the computer to display only the answer (without the word RESULT) by typing the word PRINT (or its abbreviation PR) before you type the mathematical expression. The computer displays the word RESULT only if you give it an arithmetic expression to evaluate, and don't

tell it what you want it to do with the answer. Here's how the screen would look if you used the word PRINT to tell the computer what to do with the result of $1+2+3$:

```
PRINT 1 + 2 + 3
6
```

How to Evaluate Arithmetic Expressions

When an arithmetic expression contains more than one operation symbol, the result obtained may depend on the order in which the operations are carried out. There are two ways, for example, that the expression $1+2\times3$ could be carried out. If you perform the addition first, the answer would be 9 because $1+2=3$ and $3\times3=9$. But if you perform the multiplication first, the answer would be 7 because $2\times3=6$ and $1+6=7$.

To avoid confusion when using expressions with more than one operation, mathematicians have developed a hierarchy of arithmetic operations. The rule is: Multiplication and division are carried out before addition and subtraction, moving from left to right; then addition and subtraction are carried out, again moving from left to right. So, the correct evaluation of the $1+2\times3$ example would be $2\times3=6$ and $1+6=7$, because the multiplication is performed first, then the addition.

The hierarchy of arithmetic operations is built into Logo, so the computer automatically evaluates arithmetic expressions according to this standard hierarchical order. You can verify this for yourself by asking the computer to evaluate $1+2\times3$ and seeing if it performs the multiplication before the addition to obtain the correct answer (7). Just type PRINT 1+2*3 and see what answer you get. I'll bet it's 7.

On the other hand, there may be occasions when you don't want the operations performed in the standard hierarchical order. At those times, you need some way of telling

the computer you want the expression evaluated in some other order. For example, you may want the addition in $1+2\times3$ to be performed before the multiplication. To tell the computer, or anyone else, that you want the addition done first, you enclose that part of the expression within parentheses, as in $(1+2)\times3$. When the computer reads parentheses within an arithmetic expression, it evaluates the part of the expression within the parentheses before going on to evaluate the rest of the expression. So, the result of $(1+2)\times3$ is 9, because $1+2=3$ and $3\times3=9$.

The following is a brief summary of the hierarchy of arithmetic operations.

1. If a mathematical expression contains parentheses, evaluate the part within the parentheses first, according to steps 2 and 3.
2. Perform multiplication and division first, moving from the left of the expression to the right.
3. Then, perform addition and subtraction in the expression, moving from left to right.

More Things to Do

Let's try evaluating a few mathematical expressions to practice using the hierarchy of operations. First, try evaluating the following expressions using pencil and paper. Then, enter them on the keyboard exactly as they're written here, preceded by the word PRINT and a space, to see if the computer's answer matches yours. Remember that the operation of multiplication is represented by the <*> key and division is represented by the </> key in Logo. If the two answers don't agree, go back and redo the problem to find your mistake. Since the computer automatically follows the hierarchy rules, these examples not only provide practice in using

the computer to evaluate mathematical expressions, but they also let you know whether you understand the rules well enough to use them correctly.

1. $2 \times 7 - 3$

2. $2 \times (7 - 3)$

3. $6 + 12 \div 3 - 2$

4. $(6 + 12) \div 3 - 2$

5. $(6 + 12) \div (3 - 2)$

6. $1 + 2 \times 3 + 4 \times 5 + 6$

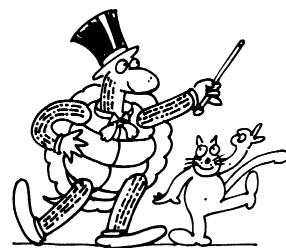
7. $(1 + 2) \times (3 + 4) \times 5 + 6$

8. $(1 + 2) \times (3 + 4) \times (5 + 6)$

Using Arithmetic Operations In Procedures

Writing and using procedures containing any of the four arithmetic operations is much the same as writing and using procedures containing graphics commands. The major difference is that whenever a procedure contains a computational statement, it must also contain a statement that tells the computer what to do with the result of the computation. If the procedure doesn't contain such a statement, the computer will interrupt the procedure and display an error message saying it doesn't know what to do with the result.

For example, in the following procedure, called COUNT.UP, when you type a positive integer (that is, a positive whole number, such as 1, 2, 3, or 4) in place of the



Turtle Tip: The hierarchy of arithmetic operations described in this chapter is a topic that's used throughout school mathematics from junior-high (and even upper-elementary) school arithmetic through secondary-school algebra and beyond. Because this hierarchy is used so much in advanced mathematics, it's important that your child practice using it so that its correct use becomes second nature.

Unfortunately, many children never master this hierarchy, and those who don't suffer the consequences throughout their school years. By encouraging the use of Logo as a learning aid at home, the same way hand-held calculators are used as learning aids in the mathematics classroom at school, you can help your child master this hierarchy.

Suppose, for example, that your child is learning to evaluate mathematical expressions and is doing his or her homework by hand, using pencil and paper. After completing the work by hand, let your child check the results on the computer. This will provide practice in using the hierarchy of operations and practice in using Logo's computational capabilities at the same time. In addition, it lets your child determine immediately whether the problem was done correctly or not. If the answer is correct, your child can go on to the next problem. But if the answer is incorrect, you and your child can go back and redo the problem, trying to discover where the error occurred.

variable :N, the procedure prints out on the display screen the positive integers beginning at 1 and ending at the number you supplied.

```
TO COUNT.UP :N
 MAKE "K 1
 REPEAT :N [PRINT :K MAKE "K :K+1]
END
```

An alternate form of COUNT.UP, called COUNT.ALT, uses the subprocedure SHOW to display the consecutive positive integers and the conditional statement IF-THEN to determine when to stop.

```
TO COUNT.ALT :N
 MAKE "K 1
 SHOW
END
```

When you use Logo with your child, make sure you use mathematical problems that are appropriate to his or her age and grade level. For example, a child in grade six or seven (age 11 or 12) should be able to solve simple problems involving only one or two of the arithmetic operations at a time. A child in grade eight or nine (age 13 or 14) should be able to solve slightly more difficult problems involving several of the arithmetic operations or simple problems with parentheses. A child in grade nine or beyond (age 15 or older) should be able to solve problems involving both arithmetic operations and parentheses in various combinations.

If you're not sure just how difficult a problem your child can understand, start with the simplest type and increase the complexity of the problems slowly. When the problems become too difficult for your child to work, that's the time to stop. You can always come back to these more difficult problems later when your child has had time to think about the hierarchy of arithmetic operations and is making use of the hierarchy in school.

The benefits of using Logo this way are twofold. First, your child gets immediate reinforcement when the answer is correct, instead of having to wait until the next school day when homework problems are reviewed. Second, your child knows immediately when the answer is incorrect and can try to discover the error while the problem and the way it was done are still fresh in his or her mind.

```
TO SHOW
 PRINT :K
 IF :K=:N THEN STOP
 MAKE "K :K+1
 SHOW
END
```

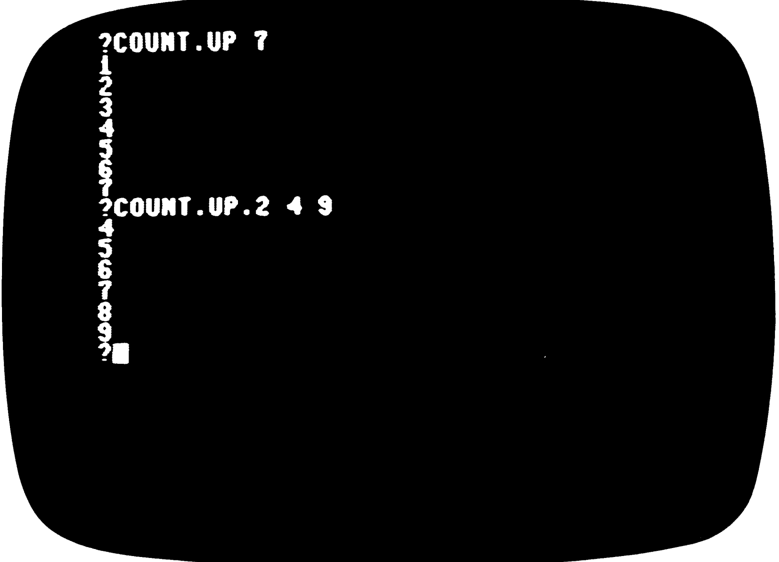
You can modify COUNT.UP and COUNT.ALT so that instead of always starting at 1 and counting up to :N, you can start at any specified positive integer and count up to any higher number. These modified procedures, called COUNT.-UP.2 and COUNT.ALT.2, take two inputs—the value :M at which the counting begins and the value :N at which the counting ends.

```
TO COUNT.UP.2 :M :N
 MAKE "K :M
 REPEAT :N-:M+1 [PRINT :K MAKE "K :K+1]
END
```

```
TO COUNT.ALT.2 :M :N
 MAKE "K :M
 SHOW.2
END

TO SHOW.2
 PRINT :K
 IF :K=:N THEN STOP
 MAKE "K :K+1
 SHOW.2
END
```

Figure 13-1 shows some screen displays you can obtain using COUNT.UP (or COUNT.ALT) and COUNT.UP.2 (or COUNT.ALT.2) in the NoDraw mode.

Figure 13-1. Sample results of COUNT.UP and COUNT.UP.2



If the counting goes too fast for your taste, you can slow it down by using a MAKE statement to make the computer count to 100 before printing the next number on the screen. The two commands that do that are:

```
MAKE "I 1
REPEAT 100 [MAKE "I :I+1]
```

Use of these two commands to slow down the printing is illustrated by the following procedures, COUNT.SLOWLY and its subprocedure SHOW.SLOWLY. Run COUNT.2 and COUNT.SLOWLY with the same values of :M and :N to see the effect of adding these commands to the procedure. Of course, you can change the length of the pause between the printing of successive numbers by using a value smaller than 100 in the REPEAT statement to reduce the pause, and a value greater than 100 to increase the pause.

```
TO COUNT.SLOWLY :M :N
 MAKE "K :M
 SHOW.SLOWLY
END

TO SHOW.SLOWLY
 PRINT :K
 IF :K=:N THEN STOP
 MAKE "K :K+1
 MAKE "I 1
 REPEAT 100 [MAKE "I :I+1]
 SHOW.SLOWLY
END
```

More Things to Do

Revise COUNT.UP and COUNT.ALT so that, instead of counting from 1 up to :N, they count from :N down to 1. Call these revised procedures COUNT.DOWN and DOWN.ALT. Then, revise COUNT.UP.2 and COUNT.ALT.2 so that, instead of counting up from :M to :N, they count down from :N to :M. Call these revised procedures COUNT.DOWN.2 and DOWN.ALT.2. Finally, revise COUNT.SLOWLY and SHOW.SLOWLY so that, instead of counting slowly from :M up to :N, they count slowly from :N back to :M. Call the revised procedures BACK.SLOWLY and SHOW.SLOWLY.2.

Turtle Tip: If you have a pre-schooler or an early-elementary-school student in your family, you can use COUNT.SLOWLY and BACK.SLOWLY to introduce your child to addition and subtraction.

Start by having your child count along out loud as the procedure COUNT.SLOWLY displays the numbers from 1 up to 10 and 1 up to 20 on your display screen. Once your child feels comfortable counting up to 10 and 20 starting at 1, change the starting value in COUNT.SLOWLY so that your child learns to count up to 10 and 20 from any number between 1 and 9.

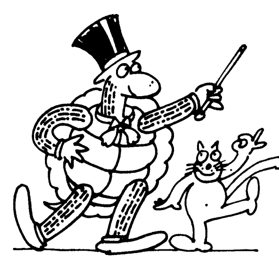
Now, give your child a simple addition problem like 5+4. Set COUNT.SLOWLY to start at 5 and count up to 20, and tell your child that 5+4 means you start at 5 and count up four more numbers. Then, run COUNT.SLOWLY 5 20 and count along as the numbers appear on the screen, saying "5 and 4 more make 6, 7, 8, 9. So, 5 plus 4 makes 9." Use COUNT.SLOWLY to help your child do other simple addition problems by starting at the first number and counting up from there until your child can do it without the procedure's help. Once your child feels comfortable doing addition this way, you can use the procedure BACK.SLOWLY to give practice in subtraction by starting at a given number and counting backward from there instead of forward.

Logo Activity Time

The TEST.ME file on your Logo Activity Disk contains a set of procedures that randomly select two 2-digit positive integers, then ask you the sum of these two numbers, and either congratulate you for giving the correct answer or show you the correct answer if your answer is wrong. The procedures are written recursively, so they'll continually generate new problems until you press <CTRL> G to stop them.

The main procedure in this file has the same name as the file itself: TEST.ME. Load the file into your workspace from your Activity Disk by typing READ "TEST.ME and then run the main procedure by typing TEST.ME.

Remember: If you want to see how these procedures are written, turn to the appendix, where chapter-by-chapter listings are given for all of the files on the Activity Disk. The only parts of the procedures you might have trouble writing now are those that involve commands not introduced yet: for



Turtle Tip: As you play TEST.ME, notice the bold plus sign beside the second number you're asked to add, the thick line underneath it, and the checkered line drawn across the screen between problems. These are just three of the graphics symbols that are produced when you press specific keys in conjunction with either the <SHIFT> key or the Commodore (<C=>) key. These symbols are shown on the front of the keys. Pressing a key in conjunction with the <SHIFT> key produces the symbol shown on the right; pressing the same key in conjunction with the Commodore key produces the symbol shown on the left. You can use these symbols in PRINT statements just as you use any of the letters, numbers, or symbols shown on the tops of the keys. (We'll discuss using PRINT to display text in more detail in Chapter 14.)

For example, the bold plus sign is obtained by pressing <SHIFT> and the <+> key; the thick line is obtained by repeatedly pressing <SHIFT> and the <*> key; and the checkered line is obtained by repeatedly pressing the Commodore key and the <+> key.

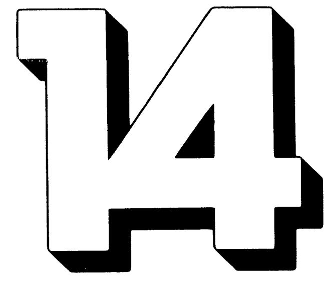
example, printing out text in the text area of the screen and using the REQUEST command to tell the computer to wait for your input on the keyboard. The REQUEST command is explained in detail in Chapter 14.

While the procedures from the TEST.ME file are in your workspace, you might find it interesting to try to modify them so they give problems in subtraction instead of problems in addition. The only subprocedures in the file that involve addition are SELECT, SHOW, and INCORRECT. Use the procedure listings in the appendix to identify the changes needed to replace addition with subtraction in these subprocedures. Then, make these changes and run TEST.ME to see if they work. Remember that the original version is still on your Activity Disk. So, if you make a mistake, you can always load that version back in again.

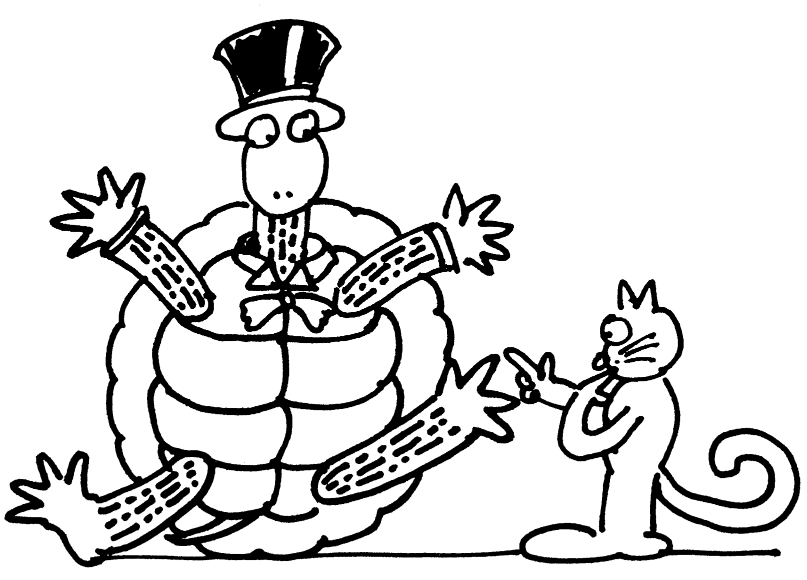
You can also try to replace the addition with multiplication or division. If you do, you might want to simplify the computations you'll have to perform by modifying the activity to use one-digit positive integers instead of two-digit positive integers. The subprocedure that randomly selects the numbers is called SELECT, so that's the subprocedure you'll have to modify if you want to change the types of numbers the activity uses.

New Commands Introduced In This Chapter

Command	*Abbreviation*
< * >	
< / >	
< + >	
< − >	
PRINT	PR



Advanced Computational Commands



In the previous chapter, you learned how to use Logo to perform arithmetic computations involving addition, subtraction, multiplication, and division. In this chapter, you'll learn Logo's more advanced computational commands and how to use these commands within procedures.

While the commands and procedures presented in the previous chapter were appropriate for children as young as 6 and 7

years of age, the commands and procedures in this chapter are more appropriate for children at the junior-high and high-school levels. In fact, some of these new commands involve trigonometry, a subject not usually taught until high school.

Let's take a look at these more advanced computational commands now.

Logo's Advanced Mathematical Commands

We'll start out by learning the names of these new commands and discussing the computational function each performs. Logo's advanced mathematics commands are:

- ROUND
- INTEGER
- QUOTIENT
- REMAINDER
- SIN
- COS
- SQRT
- RANDOM
- RANDOMIZE

The ROUND command rounds the number that follows it to the nearest whole number. For example, type:

```
PRINT ROUND 7.3
```

The screen displays the result 7. Or, type:

```
PRINT ROUND 4.85
```

The screen displays the result 5.

The INTEGER command drops any decimal part of the number that follows it, leaving only the integer part. For example, type:

```
PRINT INTEGER 3.75
```

The screen displays the result 3. Or, type:

```
PRINT INTEGER 6.2
```

The screen displays the result 6.

The QUOTIENT command divides the first of two numbers that follow it by the second and gives the integer part of the result, or quotient. For example, type:

```
PRINT QUOTIENT 9 2
```

The screen displays the result 4, since $9 \div 2 = 4.5$ and the integer part of 4.5 is 4. If decimal values are used, QUOTIENT rounds them to the nearest integer before performing the division.

The REMAINDER command divides the first of two numbers that follow it by the second and gives the remainder. (When one number is divided by another, the remainder is the decimal part of the quotient multiplied by the second number.) For example, type:

```
PRINT REMAINDER 20 3
```

The screen displays the result 2, since $20 \div 3 = 6$ with a remainder of 2 (that is, 3 divides into 20 six times, with 2 left over).

The SIN command treats the number that follows it as a number of degrees, and finds its sine. (Sine is one of the functions students learn about in high-school trigonometry. If one of the angles of a triangle is 90 degrees and another is X degrees, then the sine of X is the length of the side of the triangle opposite the X-degree angle divided by the length of the side opposite the 90-degree angle.) For example, type:

```
PRINT SIN 30
```

The screen displays the result 0.5.

The COS command treats the number that follows it as a number of degrees and finds its cosine. (Cosine is another one of the trigonometric functions students generally learn in high school. If one of the angles of a triangle is 90 degrees and another is X degrees, then the cosine of X is the length of the side of the triangle next to the X-degree angle divided by the length of the side opposite the 90-degree angle.) For example, type:

```
PRINT COS 0
```

The screen displays the result 1.

The SQRT command finds the square root of the number following it. (The square root of a number is the number that, when multiplied by itself, produces the original number.) For example, type:

```
PRINT SQRT 16
```

The screen displays the result 4 (because $4 \times 4 = 16$). Or, type:

```
PRINT SQRT 100
```

The screen displays the result 10 (because $10 \times 10 = 100$).

The RANDOM command randomly selects an integer from 0 through one less than the number you use as the input, which must be a positive integer. For example, type:

```
PRINT RANDOM 5
```

This command makes the computer pick one of the numbers 0, 1, 2, 3, or 4 at random and display it.

The problem with the RANDOM command is that it always repeats itself when the computer is turned off and then turned on again. If you turn the computer on and load Logo today and type PRINT RANDOM 5, the computer will pick and display the same number it picks and displays each time you turn the computer on, load Logo, and type PRINT RANDOM 5. So, any procedure making use of RANDOM

will repeat itself identically each time it's run. This is especially undesirable in procedures designed to randomly select numbers that must be guessed or calculated. TEST.ME, the Logo Activity Time activity at the end of the last chapter, for instance, picks two 2-digit positive integers and asks what the sum of these two numbers is. After you enter your answer, TEST.ME tells you if your answer was correct or incorrect. If TEST.ME used only RANDOM to select these numbers, the same addition problems would be selected each time you ran TEST.ME.

Fortunately, Logo provides a way of correcting this problem: the command RANDOMIZE. When RANDOMIZE is used in a procedure before RANDOM, it instructs the computer to select a different sequence of random numbers each time the procedure is run.

You'll see how RANDOMIZE is used in several of this chapter's procedures. Right now, let's begin to use the computational commands presented here to develop computational procedures. You'll learn how to use Logo's built-in computational commands to write procedures that involve both variables and recursion. Then, we'll discuss a new Logo technique: how to tell the computer to transfer information or computational results automatically from one procedure to another procedure.

Computational Procedures Using Variables

A simple but useful application of Commodore 64 Logo's built-in computational commands is the development of a procedure for finding the greatest common divisor of two positive integers. A procedure that does this, which we'll call GCD, takes two positive integers :A and :B as inputs. It uses the REMAINDER command to test every positive integer from 1 up to the smaller of the two numbers used as inputs to

determine which of the integers divide into both :A and :B evenly—in other words, which numbers are "common divisors" of :A and :B. Finally, it selects the largest of these common divisors as the "greatest" common divisor and displays that number. Here is how GCD is written:

```
TO GCD :A :B
 IF :A < :B THEN MAKE "C :A ELSE MAKE "C :B
 MAKE "D 0
 REPEAT :C [MAKE "D :D+1 IF ALLOF REMAINDER
  :A :D=0 REMAINDER :B :D=0 THEN MAKE "E :D]
 PRINT :E
END
```

If you used 2 in place of :A and 5 in place of :B in the GCD procedure, it would display 1 as the result. Using 6 and 15 would give 3 as the result.

Using GCD as a subprocedure, you can now write a procedure called ADD that takes two fractions (A/B and C/D) as inputs, adds them together in fractional form, and displays their sum in reduced fractional form—in other words, so the numerator and denominator have no divisors in common other than 1. ADD uses the following algebraic relationship to express A/B + C/D as a fraction:

$$\frac{A}{B} + \frac{C}{D} = \frac{A \times D}{B \times D} + \frac{B \times C}{B \times D}$$

$$= \frac{(A \times D + B \times C)}{B \times D}$$

Then, it expresses this fraction in reduced form by dividing both the numerator and denominator by their greatest common divisor. The main ADD procedure uses a modified version of GCD, called GCD.2, as a subprocedure.

```
TO ADD :A :B :C :D
 MAKE "J :A*:D+:B*:C
 MAKE "K :B*:D
 MAKE "L GCD.2 :J :K
```

```
  MAKE "S :J/:L
  MAKE "T :K/:L
  (PRINT :A [/] :B [ ] [+] [ ] :C [/] :D [ ] [=] [ ] :S [/] :T )
END

TO GCD.2 :J :K
  IF :J < :K THEN MAKE "U :J ELSE MAKE "U :K
  MAKE "V 0
  REPEAT :U [MAKE "V :V+1 IF ALLOF REMAINDER :J
    :V=0 REMAINDER :K :V=0 THEN MAKE "W :V]
  OUTPUT :W
END
```

Notice that while GCD computes the greatest common divisor of :A and :B and prints the result on the display screen using PRINT, GCD.2 computes the result and transfers this result to the procedure ADD using the OUTPUT command. You'll learn more about this transfer command later, but for now, just accept that it does the job needed in this case.

Try using ADD and GCD.2 now. Type:

```
ADD 1 2 1 3
```

The screen displays the result:

```
1/2+1/3=5/6
```

Now type:

```
ADD 2 6 3 10
```

The screen displays the result:

```
2/6+3/10=19/30
```

More Things to Do

Using ADD as a guide, try writing a similar procedure called SUBTRACT that evaluates and displays the difference between two fractions in reduced fractional form. Then, write

a procedure called MULTIPLY that evaluates and displays the product of two fractions in reduced fractional form. The only change required to the final PRINT statement in the main procedure will be to replace the + in ADD with − for SUBTRACT and with * for MULTIPLY. SUBTRACT and MULTIPLY will both use GCD.2 as a subprocedure, just as the ADD procedure does.

Once you've written these two new procedures, test them out using the following sample differences and products:

- Subtract 1 2 1 3: The answer should be $1/2-1/3=1/6$
- Multiply 1 2 1 3: The answer should be $1/2\times1/3=1/6$
- Subtract 4 6 3 10: The answer should be $4/6-3/10=11/30$
- Multiply 4 6 7 10: The answer should be $4/6\times7/10=7/15$

Computational Procedures That Average and Square

You can write two additional procedures that are useful to have available. We'll call them AVERAGE and SQR. AVERAGE takes two numbers as inputs and displays their average, or arithmetic mean. SQR takes one number as an input and displays its square, or the result of multiplying it by itself. Notice that AVERAGE and SQR work with decimals, fractions, and negative numbers just as easily as they work with positive integers.

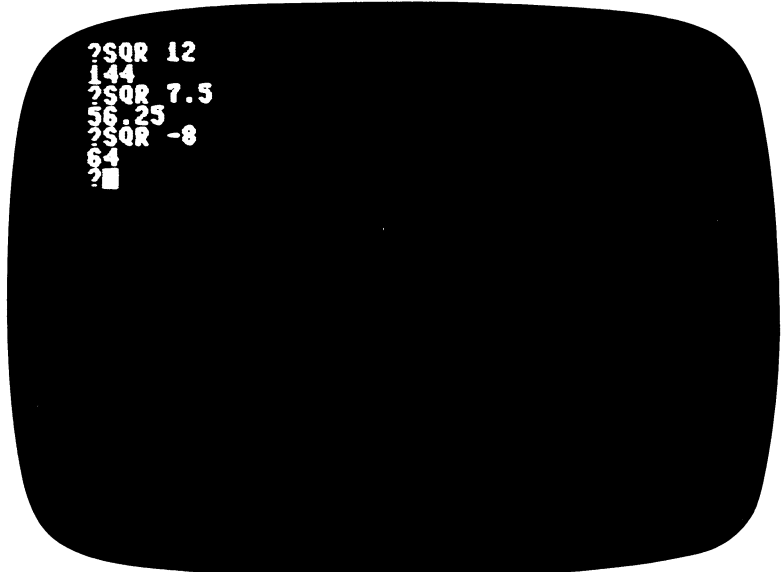
```
TO AVERAGE :FIRST :SECOND
 PRINT (:FIRST + :SECOND) / 2
END

TO SQR :NUMBER
 PRINT :NUMBER * :NUMBER
END
```

Turtle Trap: I've suggested SQR rather than SQUARE as the name of the procedure that finds the mathematical square of a number because we've already used SQUARE as the name of the graphics procedure that draws a square. If the earlier SQUARE procedure were in your workspace and you typed TO SQUARE as the first step in defining the procedure for finding the square of a number, you'd see the graphics procedure with the same name on your screen in the Edit mode. Even if the graphics procedure were saved in a file on disk, using the same name could cause a problem. If you loaded the file containing the graphics procedure while the mathematics procedure was in your workspace, the graphics procedure would replace the mathematics procedure with the same name. So, it's important not to use the same name for two different procedures, unless you want to replace an existing procedure with a newer one.

Examples of the results you can obtain using AVERAGE and SQR are shown in Figure 14-1 and Figure 14-2.

Figure 14-1. Results obtained using AVERAGE

Figure 14-2. Results obtained using SQR



More Things to Do

Using AVERAGE and SQR as guides, write a procedure that computes and displays the average of three numbers, and call it AVERAGE.3. Then, write a procedure that computes and displays the cube of a number (the number × the number × the number) and call it CUBE.

The relationship between a certain temperature measured in degrees Fahrenheit, F, and the same temperature measured in degrees Celsius, C, is $C = (5/9) \times (F - 32)$ or $F = (9/5) \times C + 32$. Write a procedure called FAH.TO.CEL that takes a Fahrenheit temperature as input, converts it to Celsius temperature, and then prints the result on the display screen. Then, write a procedure called CEL.TO.FAH that takes a Celsius temperature as input, converts it to Fahrenheit temperature, and prints the result on the display screen.

Computational Procedures Using Recursion

Instead of writing individual procedures for raising a number to the second power (SQR), to the third power (CUBE), or to any other positive-integer power, it is much more efficient to write one general procedure that works in all such cases. You can do this using the MAKE statement and recursion.

The procedure you'll want to write, which you can call EXP, uses a subprocedure called MULTIPLY and has two variable inputs. The first input, :X, represents the number to be raised to a specified power. The second input, :N, represents the power that :X is to be raised to. The power to which a number is raised is called the exponent of that number and represents the number of times that number is multiplied by itself. Try typing and running the following EXP and MULTIPLY procedures.

```
TO EXP :X :N
 MAKE "ANSWER 1
 MULTIPLY :X :N
END

TO MULTIPLY :X :N
 MAKE "ANSWER :ANSWER*:X
 IF :N=1 THEN PRINT :ANSWER TOPLEVEL
 MULTIPLY :X :N-1
END
```

The main procedure, EXP, defines the variable name "ANSWER, and gives it the starting value of 1. It then calls the MULTIPLY subprocedure, to perform all the computations and display the answer.

The subprocedure MULTIPLY multiplies the value of "ANSWER by :X. It then assigns the variable name "ANSWER this new value and calls on itself recursively to repeat these actions once again with the exponent, :N, reduced by 1.

After :N repetitions of MULTIPLY have been completed, the value of :N will have been reduced to 1, the value of "ANSWER will have become :X to the power of :N, and the conditional statement IF :N = 1 THEN PRINT :ANSWER TOPLEVEL will instruct the computer to display the result (:ANSWER) and terminate the main procedure.

Enter these two procedures and run EXP with several different pairs of numbers for :X and :N. Examples of the results you can obtain using EXP are shown in Figure 14-3. Keep in mind that you can use positive values, negative values, and decimals for :X, but the exponent :N must be a positive integer.

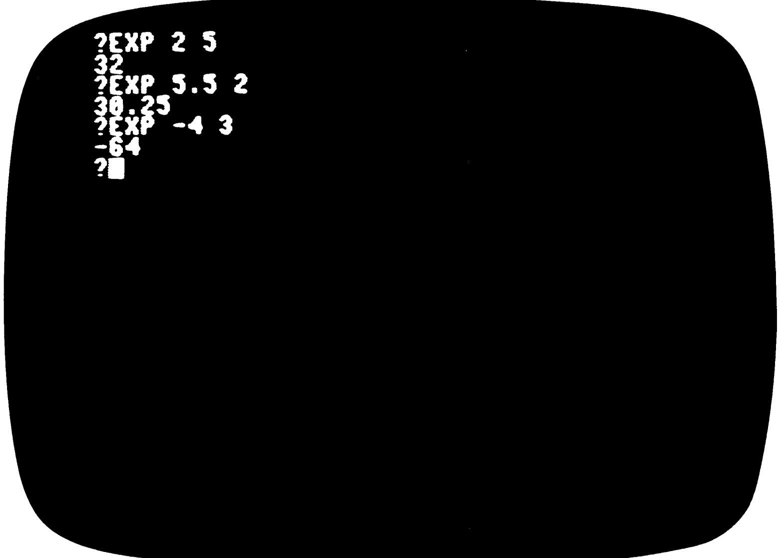


Figure 14-3. Results obtained using EXP

More Things to Do

Using EXP and MULTIPLY as guides, write a procedure (or main procedure with subprocedures) that takes a positive integer as input and prints out the sum of all the positive integers from 1 up to and including that number. That is, if

you enter 2, the procedure computes and displays 3 (since $1+2=3$) or if you enter 3, the procedure computes and displays 6 (since $1+2+3=6$). Call this procedure SUM.TO.N.

Once you've successfully written the SUM.TO.N procedure, write a similar procedure that takes any positive integer as input and prints out the product of all of the positive integers from 1 up to and including that number. So, if you enter 3, the procedure computes and displays the product 6 (since $1\times2\times3=6$) or, if you enter 5, the procedure computes and displays 120 (since $1\times2\times3\times4\times5=120$). Call this procedure MULT.TO.N.

Transferring Results Between Procedures

In the previous section, we discussed the main procedure EXP and its subprocedure MULTIPLY, which compute and display the result obtained when a specified number (:X) is raised to a specified positive-integer power (:N). With EXP and MULTIPLY, it's the subprocedure MULTIPLY that both computes the desired result and instructs the computer to print it out on the display screen.

There are other situations, however, when you don't want to display the result of a subprocedure because that result is only a partial, rather than a final, result. Instead, you want some way to transfer the partial result to the main procedure (or to some other subprocedure) so that it can be used in obtaining the final result. The command that allows you to transfer a computational result from a subprocedure to another procedure is OUTPUT (or OP for short). OUTPUT is used in the procedure from which the result is being sent.

Let's look at the use of the OUTPUT command in a procedure that uses a topic from secondary-school geometry, the Pythagorean theorem. This theorem says you can calculate the length of the longest side of a triangle with a 90-degree angle if you know the lengths of the other two sides. A triangle with a 90-degree angle is called a right triangle because of its right, or 90-degree, angle. To find the length of the longest side—called the hypotenuse—you multiply the lengths of each of the other two sides by themselves—or square them—and then add the results of the two multiplications. Then, to find the length of the hypotenuse, you have only to find which number, multiplied by itself (squared), is the same as the sum of the squares of the other two sides. Finding that number is called finding the square root. Written in a condensed form, the theorem is:

C squared = A squared + B squared

with C representing the length of the hypotenuse and A and B representing the lengths of the other two sides. That relationship also can be written as:

C = the square root of (A squared + B squared)

In Logo, you'd write :C = SQRT(:A * :A + :B * :B) to represent the same relationship.

Using this relationship, it's a fairly easy matter to write a procedure that takes numbers for :A and :B as inputs, then computes and displays the corresponding value of :C. A procedure to do this, called PYTHAG, uses the subprocedure SQR.2 as follows:

```
TO PYTHAG :A :B
 MAKE "NUM1 SQR.2 :A
 MAKE "NUM2 SQR.2 :B
 MAKE "C SQRT(:NUM1 + :NUM2)
 PRINT :C
END
```

```
TO SQR.2 :NUMBER
 OUTPUT :NUMBER * :NUMBER
END
```

Notice that the PRINT command in the SQR procedure developed earlier in this chapter is replaced by an OUTPUT command in SQR.2. The change is necessary because, while SQR was used to compute the square of a specified value and to print the result, SQR.2 is used here merely to compute the square of specified values and to transfer that partial information to PYTHAG, the procedure that asks for it.

PYTHAG uses the statement MAKE "NUM1 SQR.2 :A to obtain the square of the first input value, :A, from the subprocedure SQR.2 and to represent this result by the name "NUM1. Similarly, PYTHAG uses the statement MAKE "NUM2 SQR.2 :B to obtain the square of the second input value, :B, from SQR.2 and to represent this result by the name "NUM2. It then uses these two results to compute the corresponding value of :C and instructs the computer to print that value on the display screen. Figure 14-4 shows the results you can obtain using the PYTHAG procedure.

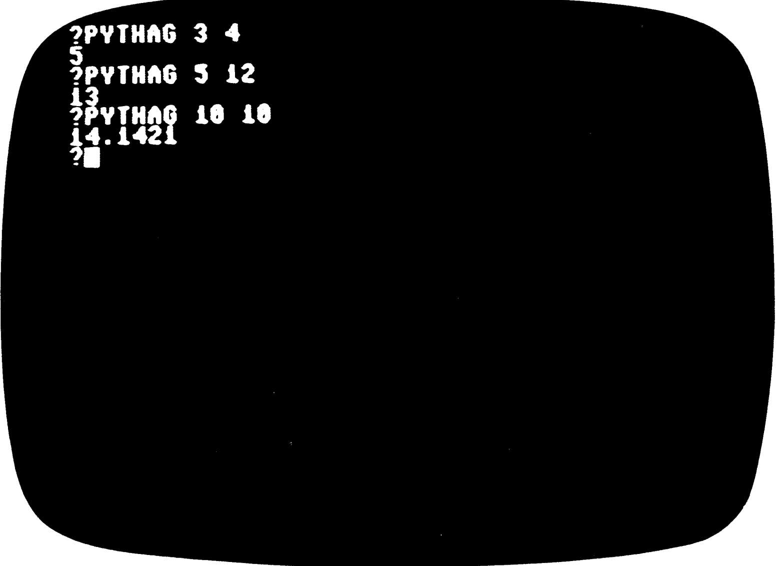


Figure 14-4. Results obtained using PYTHAG

The OUTPUT command is especially useful when a particular computation has to be performed several times in the same procedure. For example, in the last More Things to Do section, you were asked to write a procedure called MULT.TO.N that takes a positive integer as input, and then computes and displays the product of all of the positive integers from 1 up to and including that number. There's actually a mathematical function that does the same thing. It's called the factorial function and is denoted by the exclamation mark. Here are two examples of how factorial functions are written in mathematical notation:

$2! = 1 \times 2 = 2$
$3! = 1 \times 2 \times 3 = 6$

Suppose, now, that you'd like to develop a procedure that computes and displays a list of as many of these factorial values as you'd like. You could do this in a fairly compact way by writing a main procedure that keeps track of the number of factorials, determines which factorial must be computed next, and decides when to print it. Then, you could write a subprocedure that actually computes the factorial values and transfers, or outputs, those values to the main procedure. You could write a main procedure, which we'll call LIST.FAC, and its subprocedure, which we'll call FAC, to do just that. Try typing LIST.FAC and FAC.

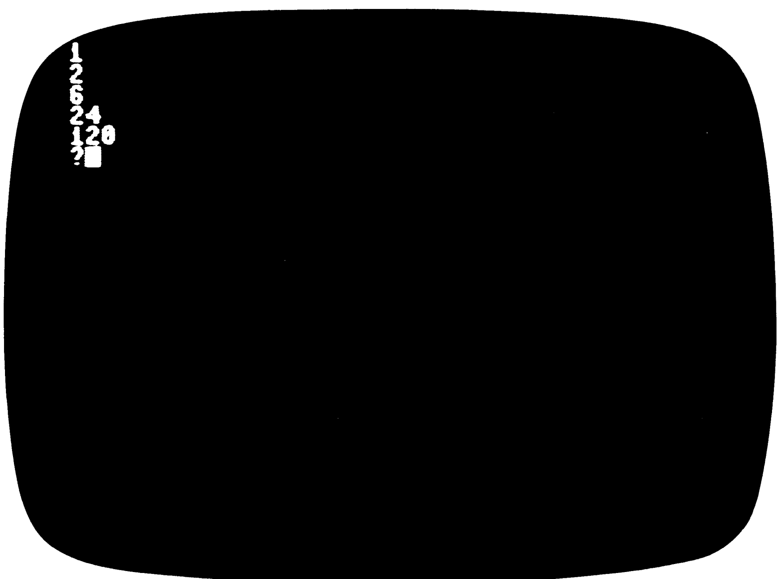
```
TO LIST.FAC :N
 ND
 MAKE "K 1
 REPEAT :N [PRINT FAC :K MAKE "K :K+1]
END

TO FAC :K
 MAKE "C 1
 MAKE "RESULT 1
 REPEAT :K [MAKE "RESULT :RESULT*:C MAKE "C :C+1]
 OUTPUT :RESULT
END
```

After defining the procedures, typing LIST.FAC 5 results in a display of the first five factorial values:

1 (1! = 1)
2 (2! = 1 × 2 = 2)
6 (3! = 1 × 2 × 3 = 6)
24 (4! = 1 × 2 × 3 × 4 = 24)
120 (5! = 1 × 2 × 3 × 4 × 5 = 120)

Each value is printed on a separate line of the screen, as shown in Figure 14-5.

Figure 14-5. Results of LIST.FAC 5



You can improve the form of this display by modifying the PRINT statement within the REPEAT statement in LIST.FAC, so that the number represented by :K and its corresponding factorial FAC :K are printed next to each other. Modify the REPEAT statement in LIST.FAC as follows:

```
REPEAT :N [(PRINT :K [] [] FAC :K ) MAKE "K :K+1]
```

The empty brackets [ ] make the computer print empty spaces between the number represented by :K and its factorial FAC :K. The parentheses ( ) enclosing the PRINT command

and the values that you want printed tell the computer the PRINT command refers to more than one number and indicate where the PRINT command ends. You'll learn more about the use of PRINT and parentheses in later chapters.

The effect of this change is evident if you now type LIST.FAC 5 again. As Figure 14-6 shows, the values of :K will now be printed to the left of, and on the same lines as, their factorial values.

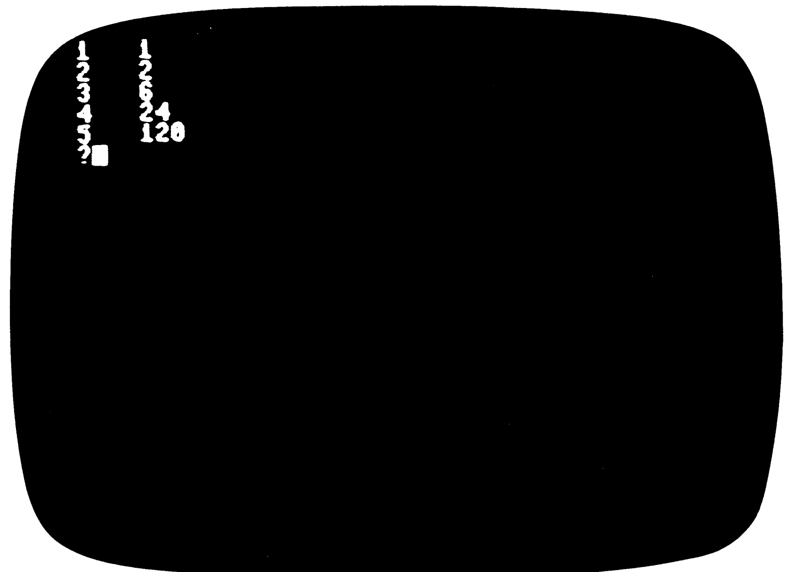


Figure 14-6. Results of modified LIST.FAC 5

More Things to Do

Write a main procedure similar to LIST.FAC and a subprocedure similar to FAC that compute and display a list of sums (results of addition) instead of products (results of multiplication). Then, modify the main procedure, as you did with LIST.FAC, so that both the number represented by :K and the corresponding sum (1 + 2 + 3 and so on) are printed out in the display.

Additional Sample Computational Procedures

In this section, we'll look at several demonstration procedures that further illustrate the use of the computational commands and built-in mathematical functions provided by Commodore 64 Logo. In the few instances where a command or technique that we haven't covered yet is used, a brief explanation of that command or technique will be supplied. More complete explanations will be given in upcoming chapters.

Guessing the Computer's Number

The following main procedure, GUESS.MY.NUMBER, randomly selects an integer between 0 and 100, and represents that number by the name "NUM using the statement MAKE "NUM RANDOM 101. It then asks you to guess the number it's picked. It asks for a guess by using the FIRST REQUEST command. You'll learn about FIRST REQUEST in the next chapter. If your guess is too high, the procedure tells you so and asks you to guess again. (You'll also learn more about printing out text on the screen like this in Chapter 15.) If your guess is too low, the procedure tells you that as well and asks for another guess. When you guess the correct value, the computer tells you and congratulates you. You can play the game over and over by typing the name of the main procedure, GUESS.MY.NUMBER.

```
TO GUESS.MY.NUMBER
 ND
 RANDOMIZE
 MAKE "NUM RANDOM 101
 PRINT [I'M THINKING OF A NUMBER BETWEEN]
 PRINT [0 AND 100. WHAT DO YOU THINK IT IS?]
 COMPARE
END
```

```
TO COMPARE
 PRINT [ ]
 MAKE "GUESS FIRST REQUEST
 IF :GUESS < :NUM THEN TOO.LOW
 IF :GUESS > :NUM THEN TOO.HIGH
 IF :GUESS=:NUM THEN YOU.WIN
 COMPARE
END

TO TOO.LOW
 PRINT [ ]
 PRINT [THAT'S TOO LOW. TRY A BIGGER NUMBER.]
END

TO TOO.HIGH
 PRINT [ ]
 PRINT [THAT'S TOO HIGH. TRY A SMALLER NUMBER.]
END

TO YOU.WIN
 PRINT [ ]
 (PRINT [THAT'S RIGHT. MY NUMBER WAS] :NUM )
 PRINT [TRY IT AGAIN. I'LL BET YOU WON'T]
 PRINT [BE SO LUCKY THIS TIME.]
 TOPLEVEL
END
```

Testing Whether a Positive Integer Is a Prime

A prime is a positive integer greater than 2 that can be divided evenly (that is, with no remainder) only by 1 and itself. For example, 6 isn't a prime since, in addition to being divisible by 1 and itself, it's also divisible by 2 and 3. The number 13 is a prime, however, since the only positive integers that divide into it evenly are 1 and itself.

Prime numbers have many interesting properties and play an important role in college-level math courses about number theory.

The following main procedure, PRIME?, and its subprocedures take a positive integer greater than 2 as input and test whether the specified number is a prime. They do this by checking to see if any of the numbers up to :N −1 divide evenly into :N. If any of them do divide evenly into :N, :N is not a prime. If none of them do, :N is a prime.

The numbers to be divided into :N are represented by the variable name :A, and the key to testing whether or not :A divides evenly into :N is the conditional statement IF INTEGER (:N/:A) = :N/:A THEN NOPRIME STOP in the FOR.NEXT.PRIME subprocedure. If one of the numbers represented by :A divides evenly into :N, then :N is not a prime. The conditional statement therefore calls the NOPRIME subprocedure, which lets you know that :N is not a prime and stops the main procedure using the TOPLEVEL command. Otherwise, if none of the numbers represented by :A divides evenly into :N, the PRIME subprocedure lets you know that :N is a prime.

The main procedure, PRIME?, and its subprocedures FOR.NEXT.PRIME, PRIME, and NOPRIME follow.

```
TO PRIME? :N
 FOR.NEXT.PRIME 2 :N
END

TO FOR.NEXT.PRIME :A :N
 IF :A = :N THEN PRIME STOP
 IF INTEGER (:N / :A) = :N / :A THEN NOPRIME STOP
 FOR.NEXT.PRIME :A + 1 :N
END

TO PRIME
 (PRINT :N [IS A PRIME] )
 PRINT [ ]
END

TO NOPRIME
 (PRINT :N [IS NOT A PRIME] )
 PRINT [ ]
END
```

Enter these procedures and run PRIME? with several different positive integers greater than 2 to see how it works. For example, typing PRIME? 3 will display the result 3 IS A PRIME, typing PRIME? 6 will display the result 6 IS NOT A PRIME, and typing PRIME? 101 will display the result 101 IS A PRIME.

Printing Out a List of Primes

The procedures just given took a specified positive integer as an input, tested it to see if it was a prime, and then displayed the result of the test. The following procedures use the same prime-testing technique, but they are different because they don't take any inputs. Instead, they consecutively test all of the positive integers larger than 2 for primality (the condition of being a prime), and print out an ever-expanding list of those numbers that are found to be primes.

```
TO PRINT.PRIMES
 ND
 CHANGE.NUMBER 3
END

TO CHANGE.NUMBER :N
 STUDY 2 :N
 CHANGE.NUMBER :N+1
END

TO STUDY :A :N
 IF :A=:N THEN (PRINT :N [IS A PRIME] ) STOP
 IF INTEGER (:N/:A)=:N/:A THEN STOP
 STUDY :A+1 :N
END
```

Enter the procedures and type PRINT.PRIMES to start the list going. The first several primes will appear fairly quickly, since they're close together and the computer doesn't need to perform too many divisions with them. But the later

additions to the list come much slower, once the number of non-primes the computer has to eliminate and divisions it has to perform begin to increase. You can stop the procedure at any time by pressing <CTRL> G.

Graphing a Trigonometric Function

The main procedure GRAPH presented here, together with its subprocedures, illustrates how Logo's computation and graphic capabilities can be integrated. GRAPH is designed to sketch a part of the graph of the trigonometric function sine S multiplied by cosine S. (Here are the definitions of sine and cosine again: Given a triangle with one angle of 90 degrees and another angle of S degrees, sine S is the length of the side opposite the S-degree angle divided by the length of the side opposite the 90-degree angle; and cosine S is the length of the side adjacent to the S-degree angle divided by the length of the side opposite the 90-degree angle.)

GRAPH takes three numbers as input, with :A as the smallest value of :S you want to use in evaluating the function and drawing its graph, :B as the largest value of :S you want to use, and :INCR as the amount by which :S is to be increased in order to give the next value of :S to be evaluated and graphed. The GRAPH procedure is intentionally simplistic, but it gives a feeling for what you can now do with the graphic and computational capabilities you've learned.

GRAPH has been designed so that the curve it draws always begins at the left edge of the screen and ends at the right edge. The curve is high enough for you to see its shape easily, but not so high that it goes off the screen. These "scaling" considerations ensure that the curve fits nicely onto the display screen and isn't difficult to draw.

Enter these procedures and run GRAPH with different values for the starting point of the curve (:A), the ending point of the curve (:B), and the increment value (:INCR).

```
TO GRAPH :A :B :INCR
 DRAW
 FULLSCREEN BG 6
 SET.AXES
 MAKE "S :A
 MAKE "X (-150)
 MAKE "Y FUNCTION :A
 PU SETXY (:X) (:Y) PD
 SKETCH :A :B :INCR
END

TO SET.AXES
 HT PC 1
 FD 120 BK 240 FD 120
 LT 90
 FD 150 BK 300 FD 150
 RT 90
END

TO SKETCH :S :B :INCR
 IF :S > :B THEN STOP
 MAKE "X (300 * (:S - :A) / (:B - :A)) - 150
 MAKE "Y 100 * FUNCTION :S
 SETXY (:X) (:Y)
 SKETCH (:S + :INCR) :B :INCR
END

TO FUNCTION :S
 OUTPUT (SIN :S) * (COS :S)
END
```

Figure 14-7 shows the curve obtained using the command GRAPH 0 360 30. The smaller the value of :INCR you choose, the more points the computer uses in drawing the curve and the smoother the curve appears. On the other hand, the more points that are used, the longer it takes the

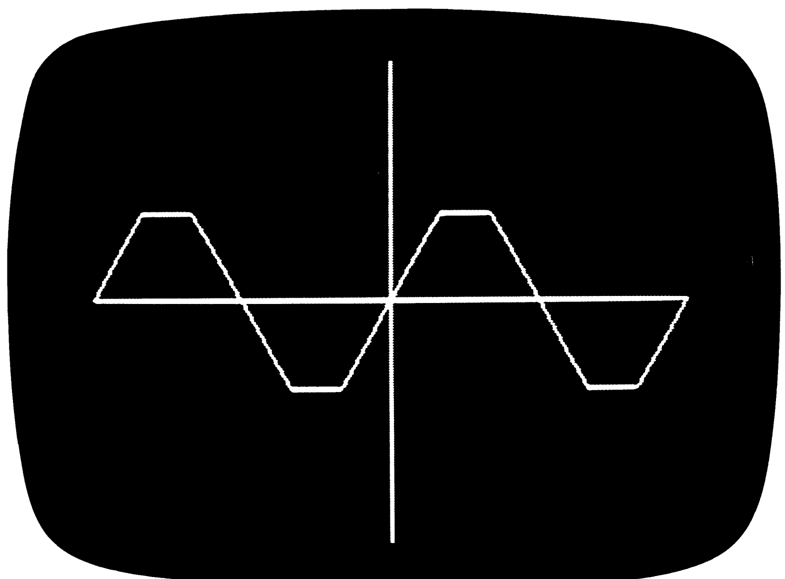


Figure 14-7. The curve obtained using GRAPH 0 360 30

computer to compute all of the function values, and the longer it takes to draw the curve. Choosing 5 for :INCR gives a much smoother and more pleasing curve than choosing 30 for :INCR, and the result is worth the extra time it takes the computer to draw the curve.

Since the function being sketched appears only in the FUNCTION subprocedure, it's a fairly simple matter to modify GRAPH for use with any trigonometric or even non-trigonometric function you might want to sketch. For example, if you simply change the multiplication symbol (*) in FUNCTION to an addition symbol (+), then GRAPH will sketch the curve of the function (SIN :S) + (COS :S) instead of the function (SIN :S) * (COS :S). The only thing you need to be careful about is that the graph of this new function doesn't extend so high or low that it goes off the top or bottom of the display screen.

The scale of the Y coordinate in GRAPH (that is, the height of the curve) is controlled by the statement MAKE "Y 100 * FUNCTION :S in the subprocedure SKETCH. If you use GRAPH with some function other than the one given here, and you find that the graph it produces is either too tall or not tall enough, you can change the scale of the Y coordinate by replacing the number 100 in the scaling statement with another number. Using a number smaller than 100 will make all of the Y coordinates proportionately smaller and the curve will look shorter; using a number larger than 100 will make all the Y coordinates proportionately larger and the curve will look taller.

Try several different numbers for the scale constant in SKETCH until you find one that gives the most pleasing appearance for your particular function. Then, keep a record of the numbers you use with each function so that you don't waste time finding the correct scale value again when you want to use GRAPH in the future.

Logo Activity Time

The BIO file on your Activity Disk uses Logo's computational capabilities and built-in mathematical functions to compute and display your biorhythm chart for any date you specify.

A biorhythm chart is composed of three characteristics that influence how you feel and how productive you are on any particular day: your physical level, your sensitivity level, and your intellectual level. People who believe in biorhythms believe a human being goes through repeated cycles of these three characteristics throughout his or her lifetime, beginning at the moment of birth. Each physical cycle takes 23 days, each sensitivity cycle takes 28 days, and each intellectual cycle takes 33 days. These cycles can be graphed, in the same way

you can graph sine curves. The curve in Figure 14-8 represents a typical physical cycle.

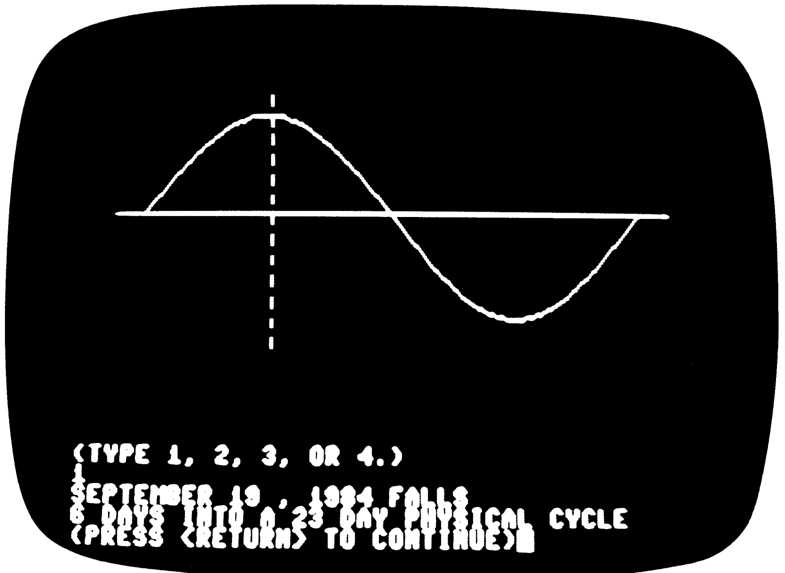


Figure 14-8. A typical physical curve produced by BIO

Each cycle of each characteristic begins at the far left of the graph at a neutral level, neither high nor low. This neutral level is represented by the number 0. According to the theory, when the cycle is at this neutral level, the person is having neither an exceptionally good time nor an exceptionally bad time for that particular characteristic. Each cycle then rises to a maximum level of the characteristic one-fourth of the way through the cycle. This maximum level is represented by a height of +1 on the curve. Each cycle then falls to a lowest level of the characteristic three-fourths of the way through the cycle. This minimum level is represented by a height of −1 on the curve. Finally, the cycle rises back to a height of 0 at the end of the cycle and is ready to immediately begin a new cycle all over again.

The two factors that determine how well you feel and how well you function at any particular point in a cycle are the height of the curve and whether the curve is rising or falling. If the curve is at a positive height, then you have a high level of the characteristic and any use of that characteristic will probably be productive. If the curve is at a negative height, then you have a low level of the characteristic and any use of that characteristic will probably be nonproductive. The worst time in a cycle is when the height is negative or zero and is still falling. This situation occurs during the third quarter of each cycle. The best time in a cycle is when the height is zero or positive and still rising. This situation occurs during the first quarter of each of the cycles.

The BIO file contains a large number of procedures, so before you read it into your workspace, you should clear the workspace by typing GOODBYE on your keyboard. (Save any procedures you want to keep first.) Then, put your Activity Disk in the disk drive, close the disk drive door, and type:

```
READ "BIO
```

The red light on the disk drive goes on and the procedures in the BIO file are read into your workspace. When the red light goes off and the cursor reappears on your screen, you know that the file has been read into your workspace and the activity is ready for your use.

Type the name of the main procedure, BIO, on your keyboard and press <RETURN>. A series of screen displays asks you for the month, day, and year of your birth and then the month, day, and year for which you want your biorhythm values computed. After you give this information, the computer computes the number of days between the two dates.

This allows the computer to determine how many complete physical, sensitivity, and intellectual cycles you've already passed through, and how many days into your current physical, sensitivity, and intellectual cycles you are.

After completing these computations, the computer displays a message asking which of the three characteristics you want to look at. (It also gives you a choice of ending the activity at this point.) Based on your response, the computer draws a graph of the appropriate cycle and draws a vertical, dotted line to indicate where in this cycle you are on the specified date. The lines of text at the bottom of the screen also provide information about your location on this cycle at the specified date.

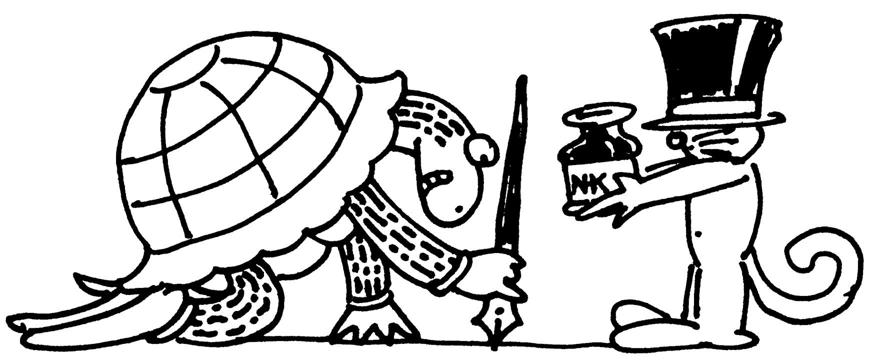
After providing this information, the computer returns you to your list of options (called the "menu" by programmers) and again asks whether you want to look at one of the cycles or leave the activity. When you want to enter a new birth date, a new biorhythm evaluation date, or both, simply select the option that takes you out of the activity (or press <CTRL> G) and type BIO again.

It's fun to take particular dates in the recent past when things went especially well or badly for you and evaluate your biorhythm values for those dates to see if the predictions match how you actually felt. Don't be disappointed if they don't match too closely, however. It's somewhat simplistic to believe that how you feel and how productive you are on any particular day is completely determined by where you are at that time in three repetitive cycles. There are obviously many other factors that influence our lives. But BIO is a fun activity and it illustrates the use of Logo's computational capabilities.

New Commands Introduced In This Chapter

Command	*Abbreviation*
ROUND	
INTEGER	
QUOTIENT	
REMAINDER	
SIN	
COS	
SQRT	
RANDOM	
RANDOMIZE	
OUTPUT	OP

15 Turning the Turtle's Pen To Prose



In addition to the commands presented so far that allow you to work with graphics and numbers, Logo also offers a number of commands that allow you to work with words. Working with words in Logo is called list processing. That might seem like an odd name, but it will seem a little less odd when you learn that a list in Logo is similar to a sentence in English grammar. Like a sentence, a Logo list is made up of words.

Words and Lists

In Logo, everything in text is either a word or a list. To the computer, a word is any sequence of one or more keyboard characters (without spaces between them). Keyboard characters can be letters, numbers, or symbols. For example, all of the following character sequences are considered words by Logo:

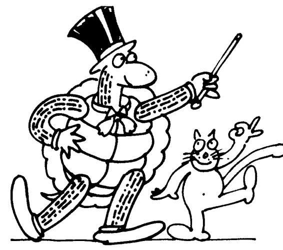
G
GOOD
1
12345
H4&

A list is a sequence of one or more of these words, with spaces between them. For example, all of the following are considered lists by Logo:

G
1
GOOD
12345
GOOD MORNING!
MY NAME IS KEN AND I HAVE TWO CHILDREN

A list can even be composed of other lists, but we'll defer discussion of that until a bit later.

Since a list can consist of just one word, it might not seem important to always distinguish between words and lists in Logo. But the distinction is important, because Logo handles words and lists differently and performs different operations with them. So, even if it makes no difference to us whether the character G or the sequence of characters GOOD represents a word or a list, it matters to Logo. We must be careful to differentiate between the two. Let's take a look at how that's done.



Turtle Tip: Children of all ages can use and enjoy the list-processing capabilities of Logo presented in this chapter. But, not everything in this chapter is appropriate for children of every age. This is one of those cases where you must decide which topics are appropriate for your child and which are not.

For example, if your child is in the first or second grade and is learning to print letters and combine them into simple words, then he or she is certainly capable of using Logo to print out individual letters and single words on the screen. If your child is in the fifth or sixth grade and writing compositions, then he or she can probably understand the Logo distinction between words and lists (especially if you talk about them as words and sentences, as you would in English). Children of this age will be able to use some of Logo's slightly more advanced list-processing capabilities to print sentences and manipulate text. If your child is in junior-high or high school, he or she should be able to incorporate Logo's list-processing capabilities into procedures to obtain even more powerful applications, such as translating foreign words and phrases into English and testing knowledge of historical events and dates. In fact, one of the procedures and the Activity Time activity in Chapter 16 do just that.

Try the topics in this chapter that you think your child can understand. If a particular topic turns out to be too difficult, simply skip it and go on to another topic. You'll find, however, that many topics that would be too difficult for your child to master on his or her own can be learned and enjoyed when the two of you work on them together.

Printing Out Words, Lists, and Variable Values

As you may recall, you used the PRINT (or PR) command in the previous chapter to tell the computer to evaluate a mathematical expression and display the result on the screen. In the same way, you can use PRINT (or PR) to tell the computer to display a word or a list on the screen. But when you use PRINT with a word or list, you need to indicate to the computer whether the text is a word or a list by using one of two symbols: an open quotation mark in front of a word or a set of brackets around a list. To tell the computer

the text is a word, you type PRINT, a space, an open quotation mark, and the word. To tell the computer the text is a list, you type PRINT, a space, an open bracket, the list, and a close bracket. You don't type a close quotation mark at the end of a word.

Suppose, for example, that you want the computer to print the word GOOD. All you have to do is put an open quotation mark at the beginning of GOOD to tell the computer that what follows is a single word. Here are several examples of the use of PRINT as they'd appear on your display screen:

```
PRINT "G
G

PRINT "GOOD
GOOD

PRINT "12345
12345
```

If you forget to put the quotation mark before G or any other word you want printed out, the computer won't know that what follows is a word and will assume that the word is the name of a procedure you want it to run. When it can't find a procedure with that name, it will assume you made a mistake and will display an error message to that effect. Here is how the mistake and error message look:

```
PRINT G
THERE IS NO PROCEDURE NAMED G

PRINT H4&
THERE IS NO PROCEDURE NAMED H4&
```

When the word contains only digits, such as 1 or 12345, the computer assumes that what it's been given is a number rather than a procedure name and will correctly print out the

number just as if it had a quotation mark, as in the following examples:

```
PRINT 1
1

PRINT 12345
12345
```

Now suppose that you want to make the computer print a list, such as GOOD MORNING! All you have to do is enclose it within brackets to tell the computer that what follows is a list. Here are several examples of this use of PRINT:

```
PRINT [G]
G

PRINT [12345]
12345

PRINT [GOOD MORNING!]
GOOD MORNING!
```

Now, just for fun, try using PRINT to tell the computer to display the message GOOD MORNING! using the open quotation mark symbol in front of only one of the two words, instead of putting brackets around both words. Here is what the display screen will show:

```
PRINT "GOOD MORNING!
GOOD
THERE IS NO PROCEDURE NAMED MORNING!
```

Can you figure out why this happened?

To begin with, since the message was preceded by a quotation mark, the computer assumed that what you wanted it to print was a single word containing no spaces. It therefore printed out just the first word in the message, GOOD. It stopped printing as soon as it reached the space following GOOD because the space indicated the word had ended.

The computer then continued reading the message to see what else you wanted it to do. When it came to the word MORNING!, it assumed that MORNING! was the name of a procedure you wanted it to run, since there were no marks to tell it differently. So, it tried to find a procedure with that name. When it couldn't find a procedure named MORNING! in your workspace, it displayed an error message to that effect and stopped to await your next command.

PRINT also can be used to make the computer display on the screen the value currently assigned to a variable name. You can do this by typing PRINT, a space, a colon, and the name of the variable. Recall that the value of a variable is represented by the variable name preceded by a colon. For example, suppose you assign the value 9999 to the variable named "BIG by using the assignment command:

```
MAKE "BIG 9999
```

You can tell the computer to display the value of "BIG using the command PRINT :BIG. Try it and you will obtain the following screen display:

```
PRINT :BIG
9999
```

Keep in mind that PRINT :BIG always prints out the current value of "BIG. So, if you change the value, the computer will print the new value rather than the old. Test this by typing:

```
MAKE "BIG 1
PRINT :BIG
```

Notice that the number printed out is the new value of "BIG (1) rather than the old (9999).

Assigning Words and Lists to Variable Names

In previous chapters, you've used variable names to represent only numerical values. But variable names can also be assigned words or lists as values. For example, the statement MAKE "SIZE "LARGE creates the variable name "SIZE and assigns it the word "LARGE as its value. If you ask the computer to print out the value of "SIZE by typing PRINT :SIZE, the computer will display the word LARGE, as follows:

```
PRINT :SIZE
LARGE
```

Similarly, the following statement creates the variable name "MESSAGE and assigns it the list [HAVE A GOOD DAY!] as its value:

```
MAKE "MESSAGE [HAVE A GOOD DAY!]
```

If you ask the computer to print out the value of "MESSAGE by typing PRINT :MESSAGE, the computer will display the list HAVE A GOOD DAY!, as follows:

```
PRINT :MESSAGE
HAVE A GOOD DAY!
```

As mentioned earlier, a list can also be composed of words and other lists. All you need to do is put a set of brackets around each sublist and a set of brackets around the main list. For example, the following statement creates the variable name "QUESTION and assigns it a list composed of two other lists:

```
MAKE "QUESTION [[HOW ARE YOU] [FEELING TODAY?]]
```

The brackets around HOW ARE YOU and FEELING TODAY? indicate to the computer that each of these sets of words is a list. The outermost brackets indicate to the computer that the two individual lists, including their brackets, are to be combined into one larger list and represented by the

variable name "QUESTION. If you ask the computer to print out the value of "QUESTION by typing PRINT :QUESTION, the computer will display everything between the outermost brackets, as follows:

```
PRINT :QUESTION
[HOW ARE YOU][FEELING TODAY?]
```

Using sublists within other lists allows you to write procedures that display or manipulate entire groups of words or numbers as individual items. For example, in Chapter 16 you'll use a procedure called TRANSLATE in which you type an English word or phrase and the computer displays the corresponding French word or phrase on the screen below it. TRANSLATE is able to do this because you've included a list of the English words and phrases and a list of their French translations within the procedure. Each English or French word or phrase is a sublist of its respective list, and is identified by a number on that list. When you use TRANSLATE, it treats the English word or phrase you type as a list and compares it to the list containing English words and phrases. If the word or phrase you typed is on the list, the computer determines its number and then selects and displays the correspondingly numbered item from the list of French translations.

You'll see several other examples of procedures that use lists composed of sublists in Chapter 16. For now, just keep in mind that there is a difference between a list composed of individual words and a list composed of those same words separated into specified sublists.

Determining the Value of a Variable Name

Since a variable name may or may not have a value assigned to it, and since a variable name can represent either a number, a word, or a list (remember that for the computer, digits can be both numbers and words), it's important to have

a way of quickly and easily determining the current status and value of any variable name. The commands that let you do that are:

- THING?
- THING
- NUMBER?
- WORD?
- LIST?

The THING? command takes a variable name as input and displays TRUE if the name has a value assigned to it or FALSE if it doesn't have a value assigned to it. For example, suppose "SIZE has a value assigned to it, but "NOTHING does not. Typing PRINT THING? "SIZE would display the result TRUE, and typing PRINT THING? "NOTHING would display the result FALSE.

The THING command takes a variable name as input and displays the value assigned to that name. For example, suppose "SIZE has the value "LARGE assigned to it and "MESSAGE has the value [HAVE A GOOD DAY!] assigned to it. Typing PRINT THING "SIZE would display the result LARGE and typing PRINT THING "MESSAGE would display the result HAVE A GOOD DAY!

The NUMBER? command takes one input and displays TRUE if the input is a number or FALSE if the input isn't a number. For example, 1234 is a number, but "HELLO is not. So, typing PRINT NUMBER? 1234 would display the result TRUE and typing PRINT NUMBER? "HELLO would display the result FALSE.

The WORD? command takes one input and displays TRUE if the input is a word or FALSE if the input isn't a word. For example, suppose "SIZE represents a word, but

"MESSAGE does not. Typing PRINT WORD? :SIZE would display the result TRUE and typing PRINT WORD? :MESSAGE would display the result FALSE.

The LIST? command takes one input and displays TRUE if the input is a list or FALSE if the input isn't a list. For example, suppose "MESSAGE represents a list, but "SIZE does not. Typing PRINT LIST? :MESSAGE would display the result TRUE and typing PRINT LIST? :SIZE would display the result FALSE.

The NUMBER?, WORD?, and LIST? commands are especially useful when you write procedures that request and use inputs while they're actually running, as we'll see later in this chapter. Such procedures are called interactive, because you interact with them the same way you receive, use, and give information when interacting with people. If a procedure is specifically written to expect and use a particular type of input, such as numbers, words, or lists, you may want to include NUMBER?, WORD?, and LIST? to test the type of input provided and an IF-THEN command to tell the computer what action to take if the input is not the correct type. This kind of input test is illustrated by the following pair of statements:

```
TEST NUMBER?
IFFALSE THEN
```

The first statement tests to see if the value following NUMBER? is a number and the second statement tells the computer what action to take if it's not. If you want to test whether a value is a word or a list rather than a number, simply replace NUMBER? with WORD? or LIST?.

While the list-processing commands you've learned in this chapter are useful in their own right, their real power comes when you use them within procedures. In the following sections, you'll see the results that can be obtained by using Logo's list-processing capabilities in procedures that use

repetition and recursion. Then, you'll learn how to create interactive procedures that can ask for, use, and display information while they're running.

A Simple Procedure That Uses Recursion to Print Text

The PRINT command can be used within a procedure to display text on the screen at an appropriate time. For example, the following procedure, which we'll call ME, uses PRINT:

```
TO ME
 PRINT [YOU'RE A WONDERFUL HUMAN BEING AND
  VERY BRIGHT, TOO.]
END
```

Switch the screen to the Text mode (if you're not already in that mode) and clear the screen by typing TEXTSCREEN and CLEARTEXT. Now, run ME and see how much your computer thinks of you.

If that acclamation isn't enough, you can make the computer repeat itself endlessly by making ME recursive. The following modified form of ME, called ME.2, does that. When your ego is finally gratified and you want to stop the procedure, just press <CTRL> G.

```
TO ME.2
 PRINT [YOU'RE A WONDERFUL HUMAN BEING AND
  VERY BRIGHT, TOO.]
 PRINT [ ]
 ME.2
END
```

The command PRINT [ ] makes the computer skip a line (print a blank line, actually) every time before it repeats ME.2. The command PRINT " would work equally well.

More Things to Do

Write a procedure that puts the screen in the Text mode, clears the screen, and prints your name five times on successive lines of the screen. Then, modify the procedure so it prints a blank line between each listing of your name. Once you've written and tested these procedures, write a procedure called LETTER that puts the screen in the Text mode, clears the screen, and prints the letter in Figure 15-1 on your screen.

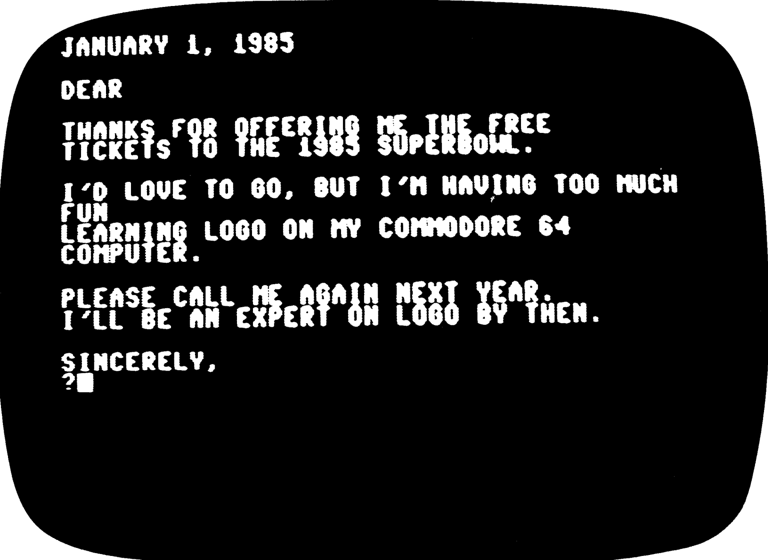


Figure 15-1. A sample letter

If you want to indent any of the text, you can do it by inserting empty brackets within the appropriate PRINT statements. For example, the statement:

```
PRINT [HOW ARE YOU]
```

displays the message HOW ARE YOU starting at the left margin of the screen. The modified statement:

```
(PRINT [] [] [] [] [] [HOW ARE YOU] )
```

prints the same message indented five spaces from the left margin of the screen. That's because each of the five pairs of empty brackets makes the computer print an empty space. The open parenthesis before the word PRINT tells the computer to print more than one item and the closing parenthesis tells the computer to stop printing. We'll discuss the use of parentheses in PRINT statements in more detail later in this chapter.

Personalizing Procedures

While it's heart-warming to be called a wonderful human being and a very bright person, the earlier message is somewhat impersonal. What you need is a way of personalizing it for whomever the computer is addressing. You do that with the REQUEST (or RQ) command. When the computer comes across this command in a procedure, it freezes the procedure that's running and waits for a message to be typed on the keyboard and the <RETURN> key to be pressed. It then treats the keyboard input as a list, whether it consists of several words, one word, or just one character.

The following modification of ME.2, which we'll call ME.3, uses REQUEST together with a MAKE command to ask the computer user for his or her name. The procedure then represents that name by the variable "NAME so that it can bestow a personally addressed compliment on the user.

```
TO ME.3
 TEXTSCREEN
 CLEARTEXT
 PRINT [HI, MY NAME IS COMMODORE 64 LOGO.]
 PRINT [WHAT'S YOURS?]
 MAKE "NAME REQUEST
 PRINT []
 (PRINT [PLEASED TO MEET YOU,] :NAME )
```

```
  PRINT [I JUST WANTED YOU TO KNOW THAT I
   THINK]
  PRINT [YOU'RE A WONDERFUL HUMAN BEING AND]
  PRINT [VERY BRIGHT, TOO.]
END
```

Since the computer user's response to REQUEST is always treated by Logo as a list (you'll learn later how to modify it so the response is treated as a word instead of a list), you can answer the computer's question with one or more words. Here are two illustrations of how the "conversation" with the ME.3 procedure might go:

```
HI, MY NAME IS COMMODORE 64 LOGO.
WHAT'S YOURS?
KEN
PLEASED TO MEET YOU, KEN.
I JUST WANTED YOU TO KNOW THAT I THINK
YOU'RE A WONDERFUL HUMAN BEING AND
VERY BRIGHT, TOO.
```

```
HI, MY NAME IS COMMODORE 64 LOGO.
WHAT'S YOURS?
KENNETH P. GOLDBERG
PLEASED TO MEET YOU, KENNETH P. GOLDBERG.
I JUST WANTED YOU TO KNOW THAT I THINK
YOU'RE A WONDERFUL HUMAN BEING AND
VERY BRIGHT, TOO.
```

Notice the parentheses in the statement:

```
(PRINT [PLEASED TO MEET YOU,] :NAME )
```

As mentioned in the previous More Things to Do section, the parentheses tell the computer that the PRINT statement applies to all of the words, lists, and variable values contained within the parentheses.

Ordinarily, a PRINT statement only applies to the one word, list, or variable value that directly follows it. In the ME.3 procedure, you wanted the PRINT statement to print

both the list PLEASED TO MEET YOU and the value of the variable "NAME on the same line. You use parentheses to indicate that to the computer. When parentheses are used to tell the computer to print more than one item, the opening parenthesis is placed just before the PRINT command and the closing parenthesis is placed just after the final item to which PRINT applies.

More Things to Do

Write a modified version of the LETTER procedure described in the previous More Things to Do section and call this new procedure LETTER.2. Write LETTER.2 so that it begins by asking you:

```
TO WHOM SHOULD THIS LETTER BE ADDRESSED?
```

Then, have it print out the letter on the display screen so that the salutation line consists of the word DEAR, a space, the name you supply, and a colon. You'll need to use parentheses around the print statement for the salutation line, since you'll want the word DEAR, the name you enter on the keyboard in response to REQUEST, and the colon to go on the same line of the display.

Additional Text-Manipulation Commands

Earlier in this chapter, you learned that Logo's list-processing capability allows you to manipulate as well as print out text. So far in this chapter, the commands you've learned have dealt primarily with printing text. The remainder of this chapter will introduce you to the text-manipulation capabilities of Logo.

Taking Selected Parts of Words and Lists

There will probably be many situations when you're using Logo's list-processing capabilities when you'll want a procedure to display or work either with the individual items of a list rather than the entire list, or with the individual characters of a word rather than the entire word.

For example, Chapter 16's TRANSLATE procedure, which we discussed earlier, compares an English word or phrase with each item of a list of words and phrases you've written into the procedure to determine if that word or phrase is on the list. If it is, the procedure locates and displays the corresponding item on the second list containing French translations.

To make this kind of item-by-item comparison, it's necessary to have list-processing commands that can separate one item of a list from all the remaining items of that list, and that can separate one character of a word from all the remaining characters of that word. Four commands that do just that are:

- FIRST
- BUTFIRST (or BF)
- LAST
- BUTLAST (or BL)

The FIRST command displays the first character of a word that follows it. When followed by a list, this command displays the first item (either a word or a sublist) of that list. For example, type:

```
PRINT FIRST "GOOD
```

The screen displays the result G. Or, type:

```
PRINT FIRST [HOW ARE YOU TODAY?]
```

The screen displays the result HOW. Or, type:

```
PRINT FIRST [[HOW ARE] [YOU TODAY?]]
```

The screen displays the result HOW ARE.

The BUTFIRST (or BF) command displays all but the first character of a word that follows it. When followed by a list, this command displays all but the first item of that list. For example, type:

```
PRINT BUTFIRST "GOOD
```

The screen displays the result OOD. Or, type:

```
PRINT BUTFIRST [HOW ARE YOU TODAY?]
```

The screen displays the result ARE YOU TODAY? Or, type:

```
PRINT BUTFIRST [[HOW ARE] [YOU TODAY?]]
```

The screen displays the result [YOU TODAY?].

The LAST command displays the last character of a word that follows it. When followed by a list, this command displays the last item (either a word or a sublist) of that list. For example, type:

```
PRINT LAST "GOOD
```

The screen displays the result D. Or, type:

```
PRINT LAST [HOW ARE YOU TODAY?]
```

The screen displays the result TODAY?. Or, type:

```
PRINT LAST [[HOW ARE] [YOU TODAY?]]
```

The screen displays the result YOU TODAY?.

The BUTLAST (or BL) command displays all but the last character of a word that follows it. When followed by a list, this command displays all but the last item of that list. For example, type:

```
PRINT BUTLAST "GOOD
```

The screen displays the result GOO. Or, type:

```
PRINT BUTLAST [HOW ARE YOU TODAY?]
```

The screen displays the result HOW ARE YOU. Or, type:

```
PRINT BUTLAST [[HOW ARE] [YOU TODAY?]]
```

The screen displays the result [HOW ARE].

One of the most useful applications of these four commands is in scanning, one at a time, the individual characters that make up a word or individual words that make up a list. This allows you to determine whether the characters or words satisfy a specified condition or match a specified item.

For example, in the next chapter you'll use a procedure called FRIEND? that asks you to type a name on the keyboard, and then compares the name you typed with a list of friends you've written into the procedure, to determine whether the name you typed is on the list and therefore a friend. As you'll see, FRIEND? and its subprocedures use the FIRST command to compare the name you type with the first item on the list of friends. If the name and the first item don't match, the BUTFIRST command is used to remove the first item from the list. FIRST is then used again to compare the name you typed with the first item on the reduced list (the second item on the original list). This process continues until either the computer matches the name you typed with one of the items on the list, or until it reaches the end of the list without finding a match. If it finds a match, the computer displays a message telling you that the name you typed is a friend. If it doesn't find a match, the computer displays a message telling you that the name you typed is not a friend.

Let's use these commands to write a simple procedure that counts the number of characters in any word you give it and displays the result. As you'll see later in this chapter, there is a built-in primitive command in Logo that does the same thing. But writing a procedure that performs this operation will illustrate how that built-in command works. It

also will provide practice in using the four commands you've just learned so that you'll be familiar with them when they're used in developing more advanced procedures in the next chapter. This procedure, called COUNT.CHAR, is given here, together with its subprocedure READ.CHAR.

```
TO COUNT.CHAR
 TEXTSCREEN
 PRINT [WHAT IS YOUR WORD?]
 MAKE "WORD FIRST REQUEST
 MAKE "NUM 0
 READ.CHAR :WORD
 (PRINT [THIS WORD HAS] :NUM [CHARACTERS.] )
 PRINT [ ]
 PRINT [ * * * * * * * * * * * * * * * * * * * * ]
 COUNT.CHAR
END

TO READ.CHAR :WORD
 IF :WORD = " THEN STOP
 MAKE "NUM :NUM + 1
 READ.CHAR BUTFIRST :WORD
END
```

Notice that it's the subprocedure, READ.CHAR, that actually counts the number of characters in the word you supply. It does that by recursively removing each of the letters in the word, one at a time, increasing the value of "NUM by one for each letter that's removed. It stops its own execution and returns control to the main procedure, COUNT.CHAR, when all of the letters in the word have been removed.

In Logo, a word from which all the letters have been removed is called an empty word and is denoted by a quotation mark without any characters following it. A key statement in READ.CHAR is the conditional command:

```
IF :WORD = " THEN STOP
```

That conditional command checks to see when all the characters have been counted and removed by comparing the current value of "WORD with the empty word " each time the procedure repeats.

Another way of testing to see if :WORD is empty is with the command EMPTY?. EMPTY? takes one input, which can be either a word or a list. It displays the result TRUE if the word or list is empty, or the result FALSE if the word or list is not empty. You could remove the conditional statement IF :WORD = " THEN STOP from READ.CHAR and replace it with the statement:

```
IF EMPTY? :WORD THEN STOP
```

Because READ.CHAR is designed to use the BUTFIRST command to remove the letters of a given word one at a time, it's essential to the correct working of READ.CHAR that the keyboard input you supply in COUNT.CHAR be treated by the computer as a word rather than a list. That's because the BUTFIRST command (and FIRST, LAST, and BUTLAST) operate differently on words than lists. When used with a word, BUTFIRST removes the first character of the word. But when used with a list, BUTFIRST removes the first item of the list. If the computer believes your input is a one-word list, BUTFIRST will remove the entire word the first time it's used, instead of removing and counting the letters that make up the word, one at a time.

Unfortunately, Logo always treats your response to a REQUEST command as a list, even if the response consists of just one word. The way you tell the computer to treat a one-word input as a word rather than as a list is to use the modified command FIRST REQUEST instead of REQUEST in the statement:

```
MAKE "WORD FIRST REQUEST
```

Since REQUEST always produces a list and FIRST always displays a word when it's followed by a list, the command

FIRST REQUEST makes the computer take the first word of your input (in this case, there is only one word) and treat it as a word instead of as a list.

Type ND or CLEARTEXT to clear the screen, and then run COUNT.CHAR. It's recursive, so it will continue running until you press <CTRL> G.

Just for fun, try typing in two words such as GO AWAY, instead of one word. Since FIRST REQUEST tells the computer to take only the first word of your input, anything after the first word will be ignored and only the characters in the first word will be counted.

As I mentioned earlier, there was really no need for you to write a procedure that counts the number of characters in a given word, since there's a primitive command that does the same thing: COUNT. COUNT works with both words and lists. If COUNT is followed by a quotation mark and then a word, it will count and display the number of characters in the word. If COUNT is followed by a list enclosed within brackets, it will count and display the number of words in the list. Here is an example of the use of COUNT with a word and with a list:

```
COUNT "TOMORROW
RESULT: 8

COUNT [TOMORROW WE'LL GO ON A PICNIC.]
RESULT: 6
```

As you learned in Chapter 13, the computer prints the word RESULT: before the answer because you didn't tell it what to do with the result of its counting. If you use the command PRINT COUNT instead of just COUNT, it will know what to do and won't have to use RESULT:, as the following examples show:

```
PRINT COUNT "HURRY
5

PRINT COUNT [LET'S NOT BE LATE.]
4
```

More Things to Do

Modify COUNT.CHAR and READ.CHAR so that each counts and displays the number of words in a specified list.

When you've successfully modified these procedures and run them a few times to make sure they work, try writing a new set of procedures that counts and displays the total number of characters in a list, including all of the words and all of the spaces. You'll probably need two subprocedures to do this—one to remove the words from the list one at a time until no words remain, and the other to count the number of characters in each word and keep a running total of all the characters plus the spaces between successive words.

Combining Words and Lists Into New Words and Lists

Two other text-manipulation commands that often come in handy are WORD and SENTENCE (or SE).

WORD takes two or more words as inputs and combines them to form one new word. But when you want WORD to combine more than two words into one word, you must put an open parenthesis in front of WORD and a space and a close parenthesis after the last word you want to combine. Here are some examples of the use of WORD with two or more inputs:

```
PRINT WORD "GOOD "BYE
GOODBYE
```

```
MAKE "COMBINE (WORD "HO "HO "HO "HO )
PRINT :COMBINE
HOHOHOHO
```

Similarly, SENTENCE (or SE) takes two or more lists as inputs and combines all of their individual elements into one new list. But, as with WORD, when you want SENTENCE to combine more than two lists into one new list, you must put

an open parenthesis in front of SENTENCE and a space and a close parenthesis after the last list you want SENTENCE to combine. Here are some examples of the use of SENTENCE with two and with more than two inputs:

```
PRINT SENTENCE [I LIKE][LOGO.]
I LIKE LOGO.

MAKE "JOIN SENTENCE [I LIKE][LOGO.]
PRINT :JOIN
I LIKE LOGO.

PRINT (SENTENCE [THIS][IS GETTING][EASIER][ALL
 THE TIME.] )
THIS IS GETTING EASIER ALL THE TIME.

MAKE "TOGETHER (SENTENCE [THIS][IS GETTING]
 [EASIER][ALL THE TIME.] )
PRINT :TOGETHER
THIS IS GETTING EASIER ALL THE TIME.
```

The following procedure, called REVERSE, takes a word as input and prints it out with its letters in reverse order. For example, if you type HELLO, REVERSE will display OLLEH. If you type BOOK, REVERSE will display KOOB. REVERSE does this by alternately using the commands LAST and BUTLAST to pick off the letters of the original word one at a time in reverse order, and the WORD command to combine these letters into a new word that's just the reverse of the original word.

Look through the REVERSE procedure to see how it makes use of the commands you've just learned. Then, enter it and run it several times to verify that it does, in fact, reverse any word you supply to it as an input value. The procedure is recursive, so it will continue to ask you for a new word to reverse until you press <CTRL> G to end it.

```
TO REVERSE
 TEXTSCREEN
 PRINT [ ]
 PRINT [WHAT WORD DO YOU WANT TO REVERSE?]
 MAKE "A FIRST REQUEST
 MAKE "B :A
 MAKE "C "
 MAKE "D COUNT :A
 REPEAT :D [MAKE "C WORD :C LAST :A MAKE "A
  BUTLAST :A]
 PRINT [ ]
 (PRINT [THE REVERSE OF] :B [IS] )
 PRINT :C
 REVERSE
END
```

More Things to Do

Using the REVERSE procedure as a guide, write a procedure that takes a list as input and prints it backward, with the last word first, the second from the last word second, the third from the last word third, and so on. Then write a procedure that takes a word as input and prints it downward, with each character of the word on a new line. Finally, write a procedure that takes a list as input and prints it downward, with each word of the list on a new line.

Logo Activity Time

The activity you'll be reading into your workspace from your Activity Disk in this chapter is called MY.WORD. It's a guessing game in which the computer randomly picks one of 10 words from two lists it has available, and asks you to guess what that word is. You have five guesses to correctly identify the word and type it in before the computer ends the activity and shows you the answer.

To help you in your guessing, the computer starts you out with one of the letters of the word filled in, and the remaining letters represented by dashes. After each incorrect guess, another of the letters is filled in to help you in your next guess. To further help you, the words in the two lists are shown here, so you can see how the activity works, both when you are able to guess a word correctly and when you make five wrong guesses.

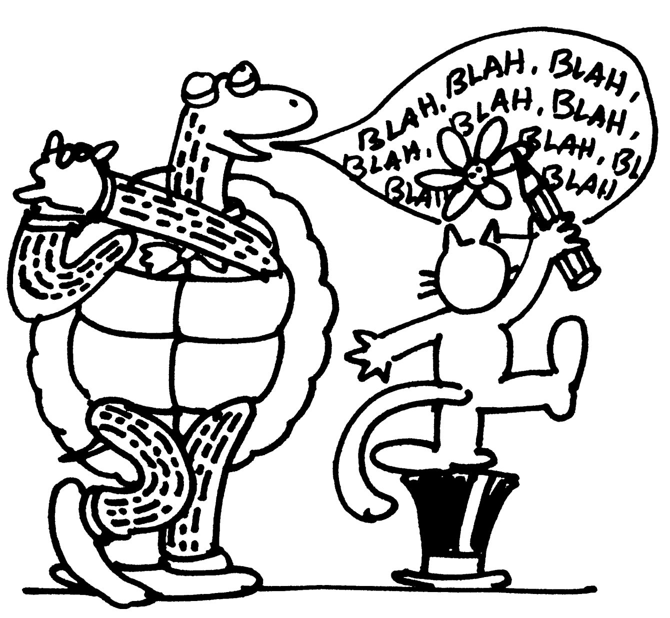
AMERICAN	BALLOONS
BIRTHDAY	CARRIAGE
COMPUTER	DOORKNOB
FOOTBALL	KANGAROO
SURPRISE	UMBRELLA

Insert your Logo Activity Disk into your disk drive and type READ "MY.WORD to load the file into your workspace. Then, type the name of the main procedure MY.WORD to begin the activity. Even though you have a list of the words from which the computer will be selecting, try purposely missing some of these words just to see how the activity works when incorrect answers are given.

New Commands Introduced In This Chapter

Command	*Abbreviation*
THING?	
THING	
NUMBER?	
WORD?	
LIST?	
REQUEST	RQ
FIRST	
BUTFIRST	BF
LAST	
BUTLAST	BL
EMPTY?	
COUNT	
WORD	
SENTENCE	SE

Activities That Speak For Themselves



Now that you know most of the built-in Logo commands for working with words, you can make your computer more interactive, rather than reactive. You'll first learn how to combine text and graphics on the screen at the same time, so you can see how Logo responds to your commands. Then, you'll learn how to use text and variables to have Logo provide different responses to your commands, depending on what those commands are.

Combining Text with Graphics

You can combine text with graphics in two ways: by using the five lines of text that appear at the bottom of the graphics screen in the SplitScreen mode, or by using the entire screen for text in the TextScreen mode. My preference, as you may have noticed from the format of the activities on the Activity Disk, is to use the SplitScreen mode while the activity is running, so that the graphics screen is always in view. I prefer using the TextScreen mode after the graphics action is completed in order to display the results of the activity and any other pertinent information.

The following procedure, called FEEDING.TIME, uses a game to show one way of combining text and graphics. The object of the game is to move the turtle close enough to its food so that it can eat. The location of the food is chosen at random by the computer, and the turtle can move only four steps at a time in any direction. You can't see where the food is, but the computer tells you how far the turtle is from the food after each move. FEEDING.TIME and its subprocedures look like this:

```
TO FEEDING.TIME
  DRAW SPLITSCREEN
  BG 6 PC 7 PU
  RANDOMIZE
  SET.LUNCH
  DISTANCE
  DISPLAY
  FIND.LUNCH
END

TO SET.LUNCH
  MAKE "X (RANDOM 301)-150
  MAKE "Y (RANDOM 151)-30
END
```

```
TO DISPLAY
 CLEARTEXT
 IF :D < 10 THEN RESULT TOPLEVEL
 (PRINT [YOU ARE AT X= ] XCOR [AND Y= ] YCOR )
 PRINT [ ]
 (PRINT [YOU ARE] :D [STEPS AWAY FROM THE FOOD.] )
END

TO FIND.LUNCH
 DISTANCE
 IF RC? THEN CHANGE RC
 FIND.LUNCH
END

TO CHANGE :CHTR
 IF :CHTR="U THEN SETH 0 FD 4 DISPLAY STOP
 IF :CHTR="D THEN SETH 180 FD 4 DISPLAY STOP
 IF :CHTR="L THEN SETH 270 FD 4 DISPLAY STOP
 IF :CHTR="R THEN SETH 90 FD 4 DISPLAY STOP
END

TO DISTANCE
 MAKE "D SQRT ((XCOR-:X) * (XCOR-:X)
   +(YCOR-:Y) * (YCOR-:Y))
END

TO RESULT
 TEXTSCREEN
 CLEARTEXT
 PRINT [CONGRATULATIONS! YOU ARE NOW WITHIN]
 (PRINT :D [STEPS OF YOUR FOOD.] )
 PRINT [ ]
 PRINT [BON APPETIT!]
END
```

When you run FEEDING.TIME, the SET.LUNCH subprocedure randomly selects a set of X,Y coordinates to serve as the location of the turtle's food. The FIND.LUNCH subprocedure lets you move the turtle four steps at a time in one of

four directions: up when you press the <U> key, down when you press the <D> key, left when you press the <L> key, and right when you press the <R> key.

DISPLAY supplies the text in FEEDING.TIME. Every time the turtle moves, DISPLAY prints the turtle's current X,Y coordinates and its distance from the food, in the text portion of the screen. This allows the player to see the turtle and its proximity to the food at the same time. Once the turtle is within 10 steps of the food, the RESULT subprocedure switches the screen to the TextScreen format and announces that the turtle is close enough to the food to eat.

Type in and run FEEDING.TIME and its subprocedures. As you play the game, notice how the display at the bottom of the graphics screen is updated every time you move the turtle, and how it switches to a full screen of text when the activity ends.

Using FEEDING.TIME as a guide, try modifying some of the graphics procedures you developed in the first seven chapters to include text as well as graphics in either the SplitScreen format, the TextScreen format, or both. You might also want to rerun some of the activities on your Activity Disk, and then look through the procedure listings for these activities in the appendix to see how the text and graphics were blended together.

More Things to Do

Revise FEEDING.TIME and its subprocedures so that, instead of displaying the turtle's distance from its lunch, the procedure just tells you whether you're moving toward the food or away from it. One way you can do this is to use variable names like NEW.DIST and OLD.DIST to represent the turtle's distance from the food before and after each move, and then display the message YOU'RE GETTING CLOSER if NEW.DIST is less than OLD.DIST or YOU'RE GETTING FARTHER AWAY if NEW.DIST is greater than OLD.DIST.

Working with Individual Items in a List

As mentioned in the previous chapter, lists are the Logo version of English sentences. Lists can consist of a single character, several characters in a row, or groups of characters separated by spaces. The power of Logo's list-processing capabilities comes from the built-in Logo commands that allow us to study and manipulate the individual items that make up a list. In this section, we'll look at some of these built-in text-manipulation commands and learn how to use them.

It's useful, when writing procedures involving lists, to be able to tell whether a specific item is part of a list and then display it. Two commands let you do just that: MEMBER? and ITEM. MEMBER? finds the item and ITEM displays it.

Finding Out Whether an Item Is in a List

MEMBER? must be followed by two inputs, each of which can be a single character, a word, or a list. If the second input is a list, the first input can be either a word or a list. If the second input is a word, the first input must be a single character. MEMBER? displays the word TRUE if the first input is contained in the second input or FALSE if it isn't. For example, try typing the following command to create the variable name "FRIENDS and assign a list of four names as its value:

```
MAKE "FRIENDS [JOHN HENRY [ELLEN BROWN]
  [MELISSA A. JOHNSON]]
```

The "FRIENDS variable represents a list composed of two words, "JOHN and "HENRY, and two sublists, [ELLEN BROWN] and [MELISSA A. JOHNSON]. If you use MEMBER? with any of these words or lists, the computer will reply TRUE (these words and sublists are contained in the list :FRIENDS). If you use any other words or lists, the

computer will reply FALSE (those words and sublists are not contained in the list :FRIENDS), as follows:

```
MEMBER? "JOHN :FRIENDS
RESULT: TRUE

MEMBER? "HENRY :FRIENDS
RESULT: TRUE

MEMBER? [ELLEN BROWN] :FRIENDS
RESULT: TRUE

MEMBER? [MELISSA A. JOHNSON] :FRIENDS
RESULT: TRUE

MEMBER? "FRED :FRIENDS
RESULT: FALSE

MEMBER? [KENNETH P. GOLDBERG] :FRIENDS
RESULT: FALSE
```

Displaying an Item in a List

ITEM takes two inputs. The first must be a positive integer, the second either a word or a list. If the second input is a word, ITEM displays the character in this word specified by the number used as the first input. If the second input is a list, ITEM displays the word or sublist specified by the number used as the first input. For example, if the second input is a word and you type:

```
ITEM 1
```

the screen will display the first character of that word. If the second input is a word and you type:

```
ITEM 2
```

the screen will display the second character of that word.

Similarly, if the second input is a list and you type:

```
ITEM 1
```

the screen will display the first word or sublist of that list. If the second input is a list and you type:

```
ITEM 2
```

the screen will display the second word or sublist of that list.

ITEM can also be combined with the FIRST, BUTFIRST (BF), LAST, and BUTLAST (BL) commands to obtain parts of any particular items in a list. The following examples illustrate the use of ITEM, both by itself and together with the FIRST, BUTFIRST, LAST, and BUTLAST commands:

```
PRINT ITEM 1 :FRIENDS
JOHN

PRINT FIRST ITEM 1 :FRIENDS
J

PRINT BUTFIRST ITEM 1 :FRIENDS
OHN

PRINT ITEM 1 "HELLO
H

PRINT ITEM 2 "HELLO
E
```

Let's write some procedures that use MEMBER? and ITEM, just to illustrate what you can accomplish with these two commands. We'll start with a simple procedure, which we'll call FRIEND?. FRIEND? asks for a name, and then compares that name to a list of the computer's own "friends." If that name is on the list, the computer displays the message:

```
OF COURSE I KNOW _____.
_____ IS A GOOD FRIEND OF MINE.
```

If that name isn't on the list, the computer displays the message:

```
NO, I DON'T KNOW _____.
_____ ISN'T ONE OF MY FRIENDS.
```

FRIEND? is recursive, so when you want to end the procedure, press <CTRL> G.

```
TO FRIEND?
 TEXTSCREEN
 CLEARTEXT
 MAKE "FRIEND.LIST [[JOHN] [HENRY] [ELLEN
  BROWN] [MELISSA A. JOHNSON]]
 BEGIN.1
END

TO BEGIN.1
 PRINT [GIVE ME A NAME AND I'LL TELL YOU]
 PRINT [IF I KNOW THAT PERSON.]
 MAKE "NAME REQUEST
 IF MEMBER? :NAME :FRIEND.LIST THEN FRIEND
  BEGIN.1 STOP
 NO.FRIEND BEGIN.1
END

TO FRIEND
 (PRINT [OF COURSE I KNOW] :NAME )
 (PRINT :NAME [IS A GOOD FRIEND OF MINE.] )
 PRINT []
 PRINT [ * * * * * * * * * * * * * * * * * * * * ]
END

TO NO.FRIEND
 (PRINT [NO, I DON'T KNOW] :NAME )
 (PRINT :NAME [ISN'T ONE OF MY FRIENDS.] )
 PRINT []
 PRINT [ * * * * * * * * * * * * * * * * * * * * ]
END
```

Enter FRIEND? and its subprocedures, then run FRIEND?. The names on this procedure's list of friends are: JOHN, HENRY, ELLEN BROWN, and MELISSA A. JOHNSON. Try using FRIEND?, both with names that are

on the list and with names that aren't, to see what the procedure does in both cases. The following are examples of screen displays you can obtain when running FRIEND?:

```
GIVE ME A NAME AND I'LL TELL YOU
IF I KNOW THAT PERSON.
ELLEN BROWN
OF COURSE I KNOW ELLEN BROWN
ELLEN BROWN IS A GOOD FRIEND OF MINE.

GIVE ME A NAME AND I'LL TELL YOU
IF I KNOW THAT PERSON.
SUPERMAN
NO, I DON'T KNOW SUPERMAN
SUPERMAN ISN'T ONE OF MY FRIENDS.
```

Let's now use both MEMBER? and ITEM in a procedure that translates English into French. We'll call it TRANSLATE. TRANSLATE will ask you to type in an English word or phrase and use the MEMBER? command to determine whether your response matches any of the items in its own list of English words and phrases. If there is a match, the procedure will determine the item number for the English word you typed in and use the ITEM command to find and print out the corresponding word or phrase from a list of French translations. If there isn't a match, we'll have the procedure display a message to that effect and stop.

Here is how TRANSLATE looks:

```
TO TRANSLATE
 TEXTSCREEN
 CLEARTEXT
 MAKE "E.LIST [[HOUSE] [TREE] [BOY] [MOTHER]
  [STRAIGHT AHEAD] [HOW ARE YOU?] [GOOD
  DAY]]
 MAKE "F.LIST [[MAISON] [ARBRE] [GARCON]
  [MERE] [TOUT DROIT] [COMMENT ALLEZ-VOUS?]
  [BONJOUR]]
 BEGIN.2
END
```

```
TO BEGIN.2
 MAKE "N 0
 PRINT [WHAT ENGLISH WORD OR PHRASE WOULD
   YOU]
 PRINT [LIKE ME TO TRANSLATE INTO FRENCH?]
 MAKE "E.PHRASE REQUEST
 IF MEMBER? :E.PHRASE :E.LIST THEN FIND.FRENCH
   :N :E.LIST BEGIN.2 STOP
 NO.FRENCH BEGIN.2
END

TO NO.FRENCH
 PRINT [I'M SORRY. I DON'T KNOW THE FRENCH]
 (PRINT [TRANSLATION FOR] :E.PHRASE )
 PRINT []
 PRINT [ * * * * * * * * * * * * * * * * * * * * ]
END

TO FIND.FRENCH :N :E.LIST
 MAKE "N :N+1
 IF :E.PHRASE=FIRST :E.LIST THEN DISPLAY.TRANS STOP
 FIND.FRENCH :N BUTFIRST :E.LIST
END

TO DISPLAY.TRANS
 (PRINT [THE FRENCH TRANSLATION OF] :E.PHRASE )
 (PRINT [IS] ITEM :N :F.LIST )
 PRINT []
 PRINT [ * * * * * * * * * * * * * * * * * * * * ]
END
```

To make it easier for you to match them, here are the English words and phrases in the main procedure's list and their French translations:

English	*French*
HOUSE	MAISON
TREE	ARBRE
BOY	GARCON
MOTHER	MERE
STRAIGHT AHEAD	TOUT DROIT
HOW ARE YOU?	COMMENT ALLEZ-VOUS?
GOOD DAY	BONJOUR

Type in and run TRANSLATE, both with words and phrases that are on the English list and with some that aren't, to see what the procedure does in each case. TRANSLATE is recursive, so when you want to stop the procedure, press <CTRL> G. The following are examples of the screen displays you can obtain using TRANSLATE:

```
WHAT ENGLISH WORD OR PHRASE WOULD YOU
LIKE ME TO TRANSLATE INTO FRENCH?
HOW ARE YOU?
THE FRENCH TRANSLATION OF HOW ARE YOU?
IS COMMENT ALLEZ-VOUS?

WHAT ENGLISH WORD OR PHRASE WOULD YOU
LIKE ME TO TRANSLATE INTO FRENCH?
GOODBYE
I'M SORRY. I DON'T KNOW THE FRENCH
TRANSLATION FOR GOODBYE
```

More Things to Do

Using FRIEND? and its subprocedures as a guide, write a set of procedures that asks for an English word as an input, and then either tells you what part of speech that word is or tells you it doesn't know its part of speech. To do this, we need to build into the set of procedures a list of nouns, a list of verbs, and a list of adjectives. We can use the variable name NOUN.LIST for the list of nouns, the variable name VERB.LIST for the list of verbs, and the variable name ADJECTIVE.LIST for the list of adjectives. Use words that are clearly only nouns, verbs, or adjectives. That way each word will go on only one list and there will be no question as to what part of speech the word is. You can then use the command MEMBER? to compare the word typed as input with each of the three lists, one at a time. If the word is on any of these lists, have the computer display a message telling you what part of speech your word is. Or, if the word is not on any of the lists, have the computer tell you it doesn't know what part of speech the word is.

Again using TRANSLATE and its subprocedures as a guide, write another set of procedures that asks you for the name of a friend or relative as an input, and then either displays that person's address and telephone number or tells you that it doesn't have that person on its list. One way to do this is to build three lists into the procedures: a list of people's names, a list of their addresses in the same order as the names, and a list of their telephone numbers, also in the same order. You can then use MEMBER? to compare a name typed as input with the list of names. If the name is on the name list, you can use ITEM to locate and display the correspondingly positioned address and telephone number from the other two lists. If the name isn't on the name list, you can have the computer display a message that the name is not on its list.

Computer-Generated Poetry

Computer-generated poetry is an interesting list-processing activity that explores the English language and its grammatical structure, and provides good practice in using Logo's list-processing capabilities.

Start by making up a simple, two-line poem (the two lines don't even have to rhyme), such as:

> The sun was shining in the sky,
> the sea was calm below.

Then, take out some of the words, leaving blanks in their places:

> The _____ was _____ in the _____,
> the _____ was _____ _____.

Now, write a procedure, or a main procedure together with several subprocedures, that randomly selects words from a list (or set of lists) that you've built into the procedures, to create a second version of this poem, with the selected words filling in the blanks.

If you use just any words to fill in the blanks, most of the poems produced will be pure nonsense. What you need to do is categorize the types of words that are needed for each blank by its part of speech, so that you can have appropriate lists of replacement words and have the computer pick from the appropriate list when it reaches a blank. For example, the first blank takes the place of the word "sun," a noun. So, the first blank should be filled only by a noun. That means you need a list of nouns. You and your child can discuss just which words are nouns and which are not, creating a list of some words that are. The same analysis can be done for each of the other blanks, so that appropriate lists are built into the procedures. Of course, if several blanks require nouns, they can all be randomly selected from the same noun list.

Turtle Tip: Children often have difficulty differentiating between parts of speech (nouns, verbs, adjectives, adverbs, prepositions, and conjunctions), or even understanding why such distinctions have to be made. The computer-generated poetry activity just presented can help your child understand and enjoy this difficult, but important, area of learning.

The study of English grammar and parts of speech often begins at the junior-high level, when children are 12 or 13 years old. So, if you have a child who is this age, or even a year or two younger, creating computer-generated poetry will fit in well with the material your child is learning in English class at school. You can also use this activity to provide a child as young as 9 or 10 years old with an early preview of English grammar before the topic is encountered in school. With the younger child, however, the activity should be treated as a game rather than as formal learning about the English language. You should also provide more guidance and give more suggestions to the younger child, since the concept of categorizing words by their function in the English language is difficult to grasp and this activity may be your child's first introduction to this concept.

When you and your child discuss categorizing words by their parts of speech, try to use standard terminology and definitions. That way, the names and definitions will be familiar to your child when they are encountered in English class. You could find these definitions in an English grammar textbook or standard dictionary, but I've included them here for your convenience.

- A noun is a word used to name a person, place, or thing (such as "man," "home," or "book").
- A verb is a word used to express action or existence (such as "run" or "is").
- An adjective is a word used to modify, or describe, a noun (such as the word "tall" in the sentence "He is a tall man").
- An adverb is a word that modifies a verb, an adjective, or another adverb (such as the word "rapidly" in the sentence "He is running rapidly").

Once the procedure is developed and appropriate lists of words built in, you can run the procedure over and over and enjoy the results of your labors and the computer's "creativity." Your child can then expand the poem to three or four lines, and create entirely new prototype poems for the computer to work from.

Procedures That Learn From Experience

One of the limitations of the procedures FRIEND? and TRANSLATE is that, while they can compare input values with the lists of items that you built into them, they can't add to these lists by themselves. However, two commands give procedures the capability to "learn" from new situations: FPUT (for PUT First) and LPUT (for PUT Last). These commands allow you to add a new item to an existing list.

The FPUT command takes two inputs, the second of which is a list, and displays a new list composed of the first input followed by the items in the list. For example, the statements:

```
MAKE "A FPUT "I [[AM MAKING][THIS LIST
 LONGER]]
PRINT :A
```

combine the word "I with the list [[AM MAKING] [THIS LIST LONGER]] to make one enlarged list with "I at the beginning. The enlarged list looks like this:

```
I [AM MAKING][THIS LIST LONGER]
```

The LPUT command takes two inputs, the second of which is a list, and displays a new list composed of the first input preceded by the items in the list. For example, the statements:

```
MAKE "B LPUT "TODAY [[HOW ARE][YOU FEELING]]
PRINT :B
```

combine the word "TODAY with the list [[HOW ARE] [YOU FEELING]] to make one enlarged list with "TODAY at the end. The enlarged list looks like this:

```
[HOW ARE][YOU FEELING] TODAY
```

Using FPUT and LPUT, you can now modify FRIEND? so that, as you run it, if it encounters a name that's not on its list, it adds the new name to the existing list. You'll need to change the NO.FRIEND subprocedure because that's the subprocedure used when a given name doesn't match an item on the list of friends. Look through the following modified version of NO.FRIEND to see how it differs from the original.

```
TO NO.FRIEND
 (PRINT [NO, I DON'T KNOW] :NAME )
 (PRINT :NAME [ISN'T ONE OF MY FRIENDS.] )
 (PRINT [BUT I WANT] :NAME [TO BE MY FRIEND] )
 PRINT [SO I'M GOING TO ADD THAT NAME]
 PRINT [TO MY LIST.]
 MAKE "FRIEND.LIST LPUT :NAME :FRIEND.LIST
 PRINT []
 PRINT [ * * * * * * * * * * * * * * * * * * * * ]
END
```

Modify the original version as indicated and run FRIEND? again to see the effect of these changes. The following sample screen display shows the way this modified procedure can now "learn" from its experiences:

```
GIVE ME A NAME AND I'LL TELL YOU
IF I KNOW THAT PERSON.
SUPERMAN
NO, I DON'T KNOW SUPERMAN
SUPERMAN ISN'T ONE OF MY FRIENDS.
BUT I WANT SUPERMAN TO BE MY FRIEND
SO I'M GOING TO ADD THAT NAME
TO MY LIST.
```

```
GIVE ME A NAME AND I'LL TELL YOU
IF I KNOW THAT PERSON
SUPERMAN
OF COURSE I KNOW SUPERMAN
SUPERMAN IS A GOOD FRIEND OF MINE.
```

More Things to Do

With the NO.FRIEND subprocedure as a guide, use FPUT or LPUT to modify the NO.FRENCH subprocedure in TRANSLATE so that whenever an English word or phrase is given that is not on the procedure's list, the procedure displays a message saying that it doesn't know the French translation, asks you what it is, and adds the English word or phrase, with its translation, to the English and French phrase lists.

After you've succeeded in modifying NO.FRENCH so that TRANSLATE can learn from its experiences, try doing the same thing with the procedures discussed in the previous section on creating computer-generated poetry. You might simply add a subprocedure that asks, each time a poem is generated, whether you'd like to add any additional words to the procedure's built-in lists. Then, if your answer is "Yes," the procedure can ask you which list you'd like to add to, using FPUT or LPUT to make the addition.

Procedures That "Remember"

There's one other modification in the self-learning FRIEND? procedure that you'll want to make. I'll describe the modification here, and then illustrate how it works in the MEMO.PAD activity in the Logo Activity Time section at the end of this chapter.

The problem with FRIEND? is that, while it keeps adding new names to its list of friends as long as you keep it running,

it "forgets" these new names as soon as you end the procedure. It "forgets" because the assignment statement (the statement that uses the MAKE command) that creates the variable name FRIEND.LIST and assigns the original four names to the list of friends is contained in the FRIEND? procedure itself. As a result, every time FRIEND? is run, this assignment statement is read by the computer and the variable name FRIEND.LIST is assigned the original list of four names again.

If you want the computer to remember the expanded version of your friend list when you stop FRIEND? and begin with the expanded list the next time you run the procedure, you have to take out the assignment statement:

```
MAKE "FRIEND.LIST [[JOHN][HENRY][ELLEN
 BROWN][MELISSA A. JOHNSON]]
```

To take it out, type EDIT FRIEND? and then edit FRIEND? so it no longer contains the assignment statement. The modified FRIEND? looks like this:

```
TO FRIEND?
 TEXTSCREEN
 CLEARTEXT
 BEGIN.1
END
```

Type <CTRL> C to define the new version of FRIEND?. Now, outside of any procedures and in ordinary NoDraw mode, type the same assignment statement and press <RETURN> to implement it in your workspace, but not as a part of your procedures. In order to add names to the list and have the computer remember them, you have to save the latest version of the list on your disk. You need to put the assignment statement in your workspace so that you can save

it in its enlarged form on your disk each time you've finished using the FRIEND? procedure, adding new names to the list of friends.

If you want to be sure that the assignment statement has been implemented, simply type PRINTOUT NAMES or PO NAMES and the name FRIEND.LIST will appear on the screen, with its original four names as sublists. If it doesn't, type the assignment statement in and press <RETURN> again and verify it once more. Now, save your workspace (the procedures, this one variable name, and its value) as a file with the name FRIEND? on your Work Disk, by placing your Work Disk into the disk drive and typing SAVE "FRIEND?.

Whenever you want to use FRIEND?, read it into your workspace from your Work Disk, run it, and when you're done, save it again with the same file name. Since any new names you've added to FRIEND.LIST will be assigned to that variable name in the workspace, you'll be saving the expanded version of FRIEND.LIST each time.

Note: When you save your workspace, you will also be saving the TRANSLATE procedure and its subprocedures. If you would rather not have it in the same file as FRIEND?, you can erase it before saving your workspace. Or, you might prefer to give the file another name to remind you that it contains TRANSLATE as well as FRIEND?.

Logo Activity Time

The activity in this section, called MEMO.PAD, is a computerized memo pad of historical events and their corresponding dates. MEMO.PAD can ask for and remember information as it runs, and will still remember this information the next time it's run.

Turtle Tip: MEMO.PAD provides an opportunity for you to encourage your child's interest in history. To do this, keep a history book, an almanac, or an encyclopedia nearby when you and your child use MEMO.PAD.

Start by looking up in the history book the four basic historical events that are built into the MEMO.PAD event list. Then, every time you want to select a historical event and corresponding date to enter into MEMO.PAD's event and date lists, you and your child can look through the history book together until you find an interesting event that you'd like to use. You can even encourage your child to look up and add events and dates on his or her own, and then quiz you on them to see if you know them.

You and your child might also want to modify MEMO.PAD so that it randomly selects an event from the event list, displays the event, and asks you the corresponding date. It can then tell you if your answer is correct or not, and, if it's not correct, display the correct answer for you. Or, you might modify it so that it randomly picks either events or dates and asks you for the corresponding piece of information. A history guessing game of this type provides a lot of fun and, at the same time, helps your child to begin to memorize important historical events and dates that will be taught in school. At the very least, using MEMO.PAD in the ways suggested here will give your child an early introduction to the availability and use of reference materials, such as the encyclopedia, that are used at the high-school and college level.

MEMO.PAD is set to begin with only the following four historical events:

1215	Signing of the Magna Carta
1492	Columbus discovers America
1776	Signing of the Declaration of Independence
1803	Louisiana Purchase

When you run the main procedure, MEMO.PAD, you'll be asked whether you want to look up a date or an event. If MEMO.PAD has the date or event you enter in its built-in date or event list, the screen displays this information for you. If MEMO.PAD doesn't have the date or event you enter, a message on the screen asks you to supply the information,

and the computer adds the date and the historical event to its date and event lists.

If, when you've finished working with MEMO.PAD, you simply clear your workspace or turn the computer off, the next time you read the MEMO.PAD file from your Activity Disk into your workspace, it will begin with the same basic four dates and events. If you want to keep the expanded list of dates and events, you'll need to save your workspace with its expanded date and event lists as a file on your Activity Disk each time you finish using MEMO.PAD. Just make sure your Activity Disk is in the disk drive and type:

```
SAVE "MEMO.PAD
```

This will save the expanded version of the activity on your Activity Disk in place of the original version, since the expanded file has the same file name as the original.

More Things to Do

Modify TRANSLATE and its subprocedures so that, just as with MEMO.PAD, TRANSLATE can learn from its experiences and remember new information the next time it's run. Do this by taking the assignment statements for E.LIST and F.LIST out of the procedures themselves and typing them into your workspace in the NoDraw mode. Then, save the file consisting of the procedures and the two assignment statements as a file named TRANSLATE on your Work Disk. As long as you always read this file into your workspace from your Work Disk whenever you want to use it, and remember to save your workspace with its expanded English and French word lists on your Work Disk when you're finished using it, TRANSLATE will remember any new words it has added.

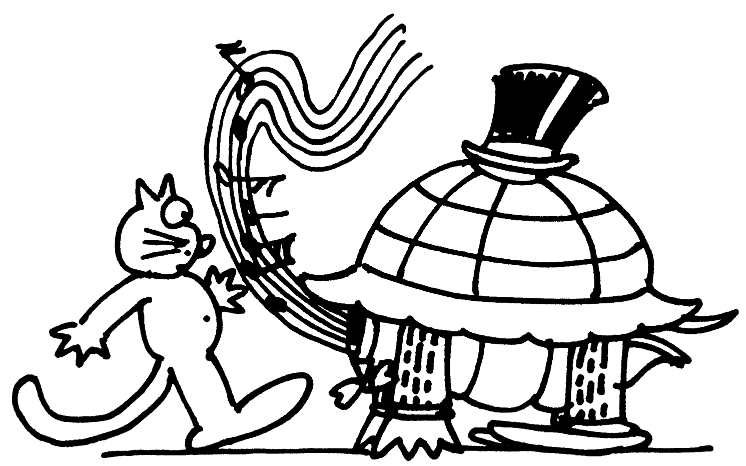
When you've successfully modifed TRANSLATE and its subprocedures as described, do the same thing with the

set of procedures discussed earlier for developing computer-generated poetry, allowing you to continually expand the choice of words from which your poems can be created without losing all of the new words each time you finish running the procedure.

New Commands Introduced In This Chapter

Command	*Abbreviation*
MEMBER?	
ITEM	
FPUT	
LPUT	

17 Sound And Music



You've done a lot in this book. You've developed graphics, computations, and text. But you haven't yet produced sounds. All of your work has been silent. In this chapter, you'll learn how to add sound effects to the procedures you've developed so far, and how to create music. Commodore 64 Logo's sound capability uses files on the Logo Utilities Disk that came with your Logo package. So, take out the Utilities Disk and insert it in your

disk drive. The Utilities Disk file you'll be using is called MUSIC. So, load this file into your workspace by typing READ "MUSIC on your keyboard.

The Characteristics of Sound

Before you start using MUSIC, you should know a little about the characteristics of sound. When a sound is generated, the three main characteristics that make it different from any other sound are its duration, its pitch, and its volume. Duration refers to how long the sound lasts; pitch refers to how high or low the sound is; and volume refers to how loud or soft it is.

When several sounds are grouped together, a fourth characteristic distinguishes that group from any other: tempo. Tempo is the speed at which the sounds occur. The MUSIC file on the Utilities Disk contains procedures that control the duration, pitch, and tempo of sounds you create with your Commodore 64. You control their volume by using the volume control knob on your monitor.

We'll begin our investigation of Logo's sound-generating capabilities by producing sounds with different durations using the SSH procedure from the MUSIC file.

Varying the Duration of a Sound

The SSH procedure takes a positive number or zero as input and produces a sound with a duration specified by the input. The larger the number, the longer the duration. The numbers don't represent a specific period of time, so you need to experiment with different numbers to find out which ones suit the sound you're trying to create.

Let's try some different numbers to hear their effects. Type the following statement, which repeats a sound with a duration of 10 and a sound with a duration of 20, five times each:

```
REPEAT 5 [SSH 10] REPEAT 5 [SSH 20]
```

If you don't hear anything when you press <RETURN>, try turning up the volume on your monitor.

The SSH command uses a predetermined pitch for the sounds it produces. Later in this chapter, you'll learn how to vary the pitch of a sound, as well as its duration.

Since 20 is twice as big as 10, the durations of the SSH 20 sounds are twice as long as those of the SSH 10 sounds. As a result, the five repetitions of SSH 20 sound slower than the five repetitions of SSH 10.

You can get a better feeling for the relationship between the number you supply as the input for SSH and the resulting duration of the sound that's produced by using the following procedure, called SLOWDOWN. SLOWDOWN uses the MAKE command to repeat each sound five times, each with durations of 0, 10, 20, and 30. Type the SLOWDOWN procedure and then run it.

```
TO SLOWDOWN
 MAKE "DUR 0
 REPEAT 4 [REPEAT 5 [SSH :DUR] MAKE "DUR :DUR+10]
END
```

Notice, when you run SLOWDOWN, how the sounds seem to slow down as the duration inputs increase.

Another procedure from the MUSIC file that's very useful for producing sounds is SSHER. SSHER takes a list of numbers as duration inputs and plays the resulting sounds one after the other. The duration values must be enclosed within parentheses and separated from each other by spaces.

Try the following SSHER statement, which produces a sound something like a drum roll:

```
SSHER [6 3 3 6 3 3 6 6 6]
```

Save this drum roll as a procedure so that you can use it together with other sound-generating procedures to produce a longer tune. The following listing for this procedure has the name ONE (since it will be used as Line 1 in a tune we'll create later in this chapter). Enter it and then try it out by typing ONE several times.

```
TO ONE
 SSHER [6 3 3 6 3 3 6 6 6]
END
```

Now, use ONE as a subprocedure, together with several other subprocedures, to produce a more complete drum roll. Let's call the main procedure DRUM and have it use the subprocedures ONE, TWO, THREE, and REST. (A rest is what musicians call the silences between successive sounds or groups of sounds.) Enter these additional subprocedures and then run DRUM to see how they sound when combined.

```
TO DRUM
 ONE REST
 ONE REST
 TWO REST
 TWO REST
 THREE
END

TO TWO
 SSHER [6 3 3 6]
END
```

```
TO THREE
 SSHER [6 3 3 6 6 6]
END

TO REST
 MAKE "COUNT 0
 REPEAT 30 [MAKE "COUNT :COUNT+1]
END
```

More Things to Do

Take a simple song you know, like "Happy Birthday" or "Twinkle, Twinkle, Little Star" and write subprocedures that play different parts of the song. Of course, at this point the only characteristic of the sounds that you can control is their duration. So, your rendition of the tune will be only a simple monotone. But go ahead and try it, just for practice. After you learn how to control the pitch of a sound, as well as its duration, you can come back to these procedures and modify them to make the tune they produce sound more like the actual song.

If you are familiar with music, you can divide the song into lines, as it would appear on a sheet of music. After you've written these subprocedures, combine them with REST in a main procedure that plays the entire tune. For example, if you're using "Twinkle, Twinkle, Little Star," the first subprocedure might play the first line "Twinkle, twinkle, little star," the second subprocedure the second line "How I wonder what you are," and so on.

Varying the Tempo of a Tune

We'll learn an easier way to change the tempo of a tune using a procedure called TEMPO on the Utilities Disk. But first, let's see how to change the tempo of a tune by changing the duration of the sounds used. We'll examine this longer

way of varying tempo first, so that you can have some additional practice in using duration values when writing sound-generating procedures. You will also become more familiar with how the TEMPO procedure works.

In the DRUM procedure, the only numbers you used as the values representing the durations of the sounds were 6 for the long notes and 3 (half of 6) for the short notes. If you replace the values 6 and 3 with two other values that are proportional to them, such as 2 and 1 or 10 and 5, you should get the same tune but at a different speed. Using 2 and 1, for example, will speed up the tune, while using 10 and 5 will slow it down.

To test this out, try the two following revised versions of DRUM and its subprocedures. DRUM.2 uses the duration values 2 and 1 in place of 6 and 3, while DRUM.3 uses duration values of 10 and 5. Enter all of these procedures and then run DRUM.2, DRUM, and DRUM.3, in that order, to go from the fastest speed to the slowest.

```
TO DRUM.2
 ONE.2 REST
 ONE.2 REST
 TWO.2 REST
 TWO.2 REST
 THREE.2
END

TO ONE.2
 SSHER [2 1 1 2 1 1 2 2 2]
END

TO TWO.2
 SSHER [2 1 1 2]
END

TO THREE.2
 SSHER [2 1 1 2 2 2]
END
```

```
TO DRUM.3
 ONE.3 REST
 ONE.3 REST
 TWO.3 REST
 TWO.3 REST
 THREE.3
END

TO ONE.3
 SSHER [10 5 5 10 5 5 10 10 10]
END

TO TWO.3
 SSHER [10 5 5 10]
END

TO THREE.3
 SSHER [10 5 5 10 10 10]
END
```

An easier way to change the speed at which a tune is played is with the TEMPO procedure found in the MUSIC file. TEMPO takes a positive number or zero as input and proportionately changes all of the sound durations based on this specified value.

When you first read the MUSIC file into your workspace from the Utilities Disk, the preset, or default, value for TEMPO is 20. Typing in an input value for TEMPO that's smaller than 20 will make all of the duration lengths shorter and, consequently, speed up all of the sounds that are produced. Typing in an input value that's greater than 20 will make all of the duration lengths longer and, consequently,

slow down all of the sounds that are produced. Enter the following statement to hear the effect produced with DRUM when the tempo is changed from 10 (fast), to 40 (slower), to 70 (slower still).

```
TEMPO 10 DRUM REST TEMPO 40 DRUM REST REST
 TEMPO 70 DRUM
```

Now, type TEMPO 20 to put the tempo back to its original default value.

Varying the Pitch of a Sound

So far, we only have a percussion section: commands that change the tempo of sounds. Now, we'll add other instruments: commands that change the pitch of the sounds. Pitch refers to how high or low a sound is.

You can vary the pitch of sounds by using the PLAY procedure. PLAY, also found in the MUSIC file, takes two lists of numbers (positive and zero) as inputs, with the first list representing the pitches and the second list representing the durations. For example, the following command generates ten sounds of increasingly higher pitch (from pitch 0 to pitch 9), but all with the same duration of 10.

```
PLAY [0 1 2 3 4 5 6 7 8 9][10 10 10 10 10 10
 10 10 10 10]
```

You can also use the letter R in place of one or more of the pitch values in the PLAY procedure. The letter R stands for rest and makes the computer remain silent for the duration corresponding to R. The letter R is a subprocedure of PLAY and is also in the MUSIC file. It serves the same purpose within PLAY as the REST subprocedure served in the DRUM procedure.

In musical notation, pitch 0 corresponds to a C note. Each increase of 1 in pitch corresponds to an increase of half

a step in the musical scale, as illustrated in Figure 17-1. (The # symbol represents a sharp, which is a half note higher, in musical notation.)

Figure 17-1. The notes produced by pitch 0 through pitch 12

Pitch	Note
0	C
1	C#
2	D
3	D#
4	E
5	F
6	F#
7	G
8	G#
9	A
10	A#
11	B
12	C

The C Major scale, a very pleasing musical progression that most people would recognize, is obtained by using just the notes on the chart that don't have sharp symbols. Figure 17-2 shows the pitches that correspond to the C Major scale.

Figure 17-2. The pitches that correspond to the C Major scale

Pitch	Note
0	C
2	D
4	E
5	F
7	G
9	A
11	B
12	C

Turtle Trap: Since every sound produced by PLAY has both pitch (the first list of input numbers) and a corresponding duration (the second list of input numbers), the second list cannot be shorter than the first. If the second list is shorter, as soon as the computer reads a pitch value and cannot find a corresponding duration value, it will immediately halt whatever procedure is running and display an error message. If the second list is longer than the first, however, the computer will read as many duration input numbers as there are pitch input numbers and then simply ignore all of the extra duration values and continue on to the next statement in the procedure.

The following procedure, which we'll call SCALE, plays C Major from the lowest pitch to the highest and then back down to the lowest:

```
TO SCALE
 PLAY [0 2 4 5 7 9 11 12][10 10 10 10 10 10 10 10]
 PLAY [12 11 9 7 5 4 2 0][10 10 10 10 10 10 10 10]
END
```

After entering and running this procedure, try speeding up the scale by using the TEMPO procedure with input values less than 20. Then, slow it down by using TEMPO with input values greater than 20. A tempo value of 10, for example, gives the impression of a person practicing the scale on a piano. When you're finished, remember to type TEMPO 20 if you want to set the tempo back to its original default value.

Creating a Simple Song

Now that you know how to vary the pitch of sounds, you can write procedures that will play recognizable songs. A very useful procedure I've developed for doing this is the COMPOSE procedure.

```
TO COMPOSE
 PLAY (LIST FIRST REQUEST)[10]
 COMPOSE
END
```

This procedure allows you to try out notes with different pitches and immediately hear how they sound. You can write down the pitch numbers that give you the right sounds for the song you're writing and use this list of pitch numbers when you write the final procedure for the song. Type in and run COMPOSE. When you press any of the number keys, followed by <RETURN>, you'll hear the corresponding note. You can even use numbers with more than one digit, such as 10, 11, and so on. COMPOSE is recursive, so when you want to stop it, press <CTRL> G. Practice with it to see how easy it is to use.

Notice that COMPOSE plays all of the pitches with a duration of 10. If you want a note with a shorter duration, replace 10 with a smaller number. If you want a note with a longer duration, replace 10 with a larger number.

I developed the following procedure, called SONG, by using COMPOSE to identify and list the correct pitches for a well-known song. I then used the pitch numbers in the PLAY procedure. Enter SONG and its subprocedures SONG.2, SONG.3, and SONG.4; and then run SONG to see if you can recognize the tune it plays. SONG.2, SONG.3, and SONG.4 play the different lines of the song. Of course, it won't sound exactly right, since all the pitches have been given a duration of 10 for simplicity. Still, you should be able to recognize it, even in this form.

```
TO SONG
 SONG.2 REST
 SONG.3 REST
 SONG.2 REST
 SONG.4
END

TO SONG.2
 PLAY [6 4 2 6 4 2 14 11 14] [10 10 10 10 10 10 10 10 10]
END
```

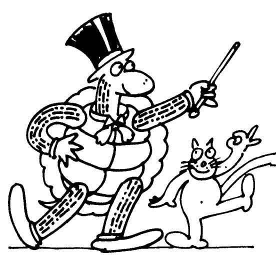
```
TO SONG.3
 PLAY [9 6 2 4] [10 10 10 10]
END

TO SONG.4
 PLAY [9 6 2 4 4 2] [10 10 10 10 10 10]
END
```

Try modifying SONG by varying the duration values within SONG.2, SONG.3, and SONG.4 as follows:

```
TO SONG.2
 PLAY [6 4 2 6 4 2 14 11 14] [30 10 5 10 5 20 20 10 20]
END

TO SONG.3
 PLAY [9 6 2 4] [30 20 20 30]
END

TO SONG.4
 PLAY [9 6 2 4 4 2] [30 10 10 30 30 30]
END
```

Make the indicated changes in the duration values and then run SONG. If you didn't recognize the tune before, you probably will now. In case you still don't recognize it, it's "Swanee River." You might also run SONG with different tempos to find one you prefer. I think a tempo of 15 or 10 gives it just the right speed. Do you agree?

You can save the procedures you've developed here, or any others, as a file on your Work Disk. Then, you can read them back into the workspace to play or modify them any time you want. Writing procedures to play music is usually hard work and very time-consuming, so don't turn off your computer or clear your workspace until you've saved any procedures you might want to use later.



Turtle Tip: The sound-generating capabilities of Logo presented in this chapter can be used by children at a variety of ages and with a wide range of musical knowledge and skills. For example, without knowing music theory and notation, a child as young as 4 or 5 can use the COMPOSE procedure to produce individual sounds and simple tunes. An older child should be able to use COMPOSE to write more complex musical procedures.

But Logo also provides an opportunity for you to help your child learn the rudiments of music theory and notation. You may want to go to your local library or bookstore and pick up an introductory book on this subject. Then, as you proceed through this chapter, you can relate the concepts covered here to the ideas presented in that book.

For example, the material on varying the duration of a sound presented earlier in this chapter can be used to explain and illustrate terms such as whole note and half note. (In a musical composition, "whole note" refers to a duration value chosen to serve as a unit of measurement against which all other durations in the composition can be compared. Once the duration for a whole note has been chosen, all the other durations in the composition are given names based on their duration relative to the whole note. Consequently, a sound that has a duration half the length of the whole note is called a half note, a sound that has a duration one-quarter the length of the whole note is called a quarter note, and so on.)

You could illustrate these two terms with the DRUM procedure, which uses duration values of 6 and 3. After running DRUM, you can explain to your child that since the number 3 is half the number 6, a sound produced using duration number 3 lasts only half as long as a sound produced using duration number 6. So, if you think of the sound produced using duration number 6 as a whole note, then it's reasonable to think of the sound produced using duration number 3 as a half note.

Similarly, when you and your child use COMPOSE to develop other procedures that play tunes, you can show your child how to record both the duration and the pitch of each sound by representing the sound as a note on a musical staff.

As we mentioned at the beginning of this chapter, Logo's sound-generating capability has many applications. It can be used for nothing more than adding sound effects to procedures you develop, or it can be used, as discussed here, to introduce your child to musical theory and notation. It's up to you which of these applications to pursue with your child and then to work with your child in exploring them.

More Things to Do

Select a song you like and write a procedure that will play it. Start by using the COMPOSE procedure to identify and make a list of the pitch numbers that give the correct notes for your song. Then, use these pitch numbers and a constant duration of 10 in the PLAY procedure to get a monotone rendition of the song. Now, try changing the duration values in PLAY until both the pitch and duration sound right. Finally, try different values for the tempo until you find one that you think plays the song at the appropriate speed. Do this with two or three of your favorite songs and save the procedures you develop as a file on your Work Disk.

Modifying Sounds You've Created

Commodore 64 Logo lets you modify the sounds you generate in four ways: by changing the waveform; by changing the attack and decay rate; by changing the sustain and release rate; and by changing the pulse. Let's take a brief look at each one.

Waveform

The term waveform refers to the quality of a sound. For example, it's the waveform that helps make the sound produced by a bell different from the sound produced by a trumpet, even when both sounds have the same duration and the same pitch.

Logo gives you a choice of four waveforms:

Waveform	*Number*
Triangle	17
Sawtooth	33
Pulse	65
Noise	129

The bell-like tones you've been producing in your songwriting have been in the Triangle waveform (Waveform 17). This waveform is the built-in, or default, value and is the one you're automatically given when you first read the MUSIC file from the Utilities Disk into your workspace. You should also be familiar with Waveform 129 (Noise)—that's the waveform used automatically with the SSH and SSHER procedures.

To switch from one waveform to another, type WAVE, a space, and the number of the waveform to which you want to switch. For example, if you'd like to hear what SONG sounds like in Waveform 33 (Sawtooth), type:

```
WAVE 33 SONG
```

If you try this, you'll find that Waveform 33 gives a sound something like a banjo, not a bad choice for the tune that SONG plays. Once you switch to a new waveform, you'll remain with that waveform until you turn the computer off, clear your workspace, or use WAVE to switch to another waveform. If you want to go back to the default waveform (Waveform 17), you'll have to do it by typing WAVE 17.

Use the WAVE command to switch to the remaining two waveforms: Pulse (Waveform 65) and Noise (Waveform 129). Run SONG in each of these waveforms to see how different the same tune can sound.

Try to choose a waveform that's appropriate for the type of sound and effect you're trying to produce. After all, a song needs a very different type of sound than the sound that's produced by an automobile crash in a game procedure. The waveform that's right for one situation might not be right for another.

Once you've decided on the waveform that's appropriate for the SONG procedure you're developing, include WAVE with the appropriate waveform number as a part of the procedure itself. That way, the procedure will call its own waveform

every time the procedure is run. You can even end the procedure with the command WAVE 17, so that Logo is put back in its default waveform as soon as the procedure is completed.

Now that you know how to change the quality of a sound, let's see how you change the speed with which a sound rises to, and falls from, its peak.

Attack and Decay

Attack refers to how fast the volume of a sound rises to its peak. Decay refers to how fast the volume of a sound falls from this peak to a level approximately midway between the peak and silence. You can change the attack speed by typing ATTACK, a space, and a number from 0 through 15. You can change the decay speed by typing DECAY, a space, and a number from 0 through 15. The larger the numbers used with ATTACK and DECAY, the longer the sound will take to reach its peak volume or fall from its peak volume. Notice that changing the attack and decay values doesn't change the duration of sounds, only how quickly during the duration the peak is reached.

Run SONG with different combinations of attack and decay values to compare the effects these combinations produce. The following are several combinations that give a good cross section of the results that can be achieved using these commands:

```
ATTACK 0 DECAY 0 SONG
ATTACK 0 DECAY 5 SONG
ATTACK 0 DECAY 10 SONG
ATTACK 5 DECAY 0 SONG
ATTACK 5 DECAY 5 SONG
ATTACK 5 DECAY 10 SONG
ATTACK 10 DECAY 0 SONG
ATTACK 10 DECAY 5 SONG
ATTACK 10 DECAY 10 SONG
```

Sustain and Release

After a sound reaches its peak, it decreases and disappears in two steps. The first step is the decay, just discussed, which takes the sound from its peak to a level approximately midway between its peak and zero. This midpoint is called the sustain level. The rate at which the sound decreases from the sustain level to zero and disappears is called the release.

You can vary both the sustain level and the release of the sounds you generate using the commands SUSTAIN and RELEASE. To change the sustain level, type SUSTAIN, a space, and a number from 0 through 15. To change the release, type RELEASE, a space, and a number from 0 through 15.

Set Logo to Waveform 17, Attack 0, and Decay 10, then try SONG or some other music-generating procedure, with several different combinations of sustain and release values. As with attack and decay, the values 0, 5, and 10 for sustain and release provide a good cross section of the effects that can be produced by varying these two characteristics.

Pulse

The term pulse refers to the richness of a sound—that is, whether the sound is tinny or full. The pulse value will only affect the sound of Waveform 65 (Pulse). The command for changing the pulse is PULSE, a space, and a number from 0 through 2048. The higher the value of pulse, the fuller and richer the sound will be. Try typing WAVE 65, so that the effects of varying the pulse can be heard, and then run SONG with several different input values, such as 0, 50, 100, 250, and 500.

Logo Activity Time

Earlier in this chapter, we focused our attention on the use of Logo's sound-generating capabilities to compose and play music. You can also use the sound capabilities of Logo to add sound effects to the kinds of activities you've been developing throughout this book. Let's now see how this can be done using an activity from your Logo Activity Disk, called WRONG.WAY.

Make sure your Logo Activity Disk is in your disk drive. Then, all you have to do is type READ "WRONG.WAY, and the activity starts itself. The screen is cleared, and you are prompted to put the Utilities Disk in the disk drive, because WRONG.WAY makes use of three of its files: MUSIC, SPRITES, and VEHICLES. After you insert the Utilities Disk and press <RETURN>, these three files are read into the computer and the main procedure, WRONG.WAY, begins.

In this activity, you're riding a bicycle from the right side of the screen to the left side of the screen on a purple street bounded by white sidewalks. You're five miles from home. The length of the part of the street you can see on the screen is one mile. So, to get home, you need to ride the length of the screen five times.

Unfortunately, the street on which you're traveling is a one-way street and you're going in the wrong direction. Consequently, each time you travel across the screen you're going to have to dodge three cars and three trucks approaching from the opposite direction. You can avoid the traffic by pressing the <L> key to move left, the <U> key to move up, and the <D> key to move down. To remind you of your choices, a list is displayed at the bottom of the screen. In addition, the bottom part of the screen keeps a count of the number of times you've been hit by a car or truck and the miles you've traveled. These counts are automatically updated each time you are hit or each time you successfully complete another trip across the screen. When you complete the five-mile journey, the activity ends and you're told how many

times you were hit during your ride. If you want to try the activity again, just type WRONG.WAY. You can stop the activity at any time by pressing <CTRL> G, but if you do, you'll need to run the ERASE.SPRITES procedure from Chapter 11 to clear the screen of all the sprites and return communication to the turtle.

You'll notice as you play WRONG.WAY that three different types of sound are included as a part of the action: the sound of a bell (in the Triangle waveform) every time a car or truck hits the sidewalk and bounces back toward the middle of the street; the sound of a crash (in the Noise waveform) every time one vehicle hits another vehicle; and the sound of your bicycle horn (in the Sawtooth waveform) each time you are hit.

After you've played WRONG.WAY a few times, turn to the appendix and look through the listings for WRONG.WAY and its subprocedures to see how the sound, animation, and list-processing capabilities were integrated to produce the final result. Then, go back to any of the other activities on the Activity Disk that you've read in and played, and try to add sound effects or a musical background. For example, you might revise the TEST.ME activity to play a stirring military tune whenever you answer a computational problem correctly. Or, you might revise the SPACETURTLE activity to play the theme from your favorite space movie as you fly your turtle spaceship across the display screen. Or, you might add sound effects to MAZE, so that you can hear a "pad, pad, pad" as the turtle moves through the maze and a "bonk" when it hits one of the walls of the maze and bounces back.

Try adding sound effects to these ready-made activities to begin with, and then move on to use sound in activities of your own design and creation.

Your Turtle Awaits You

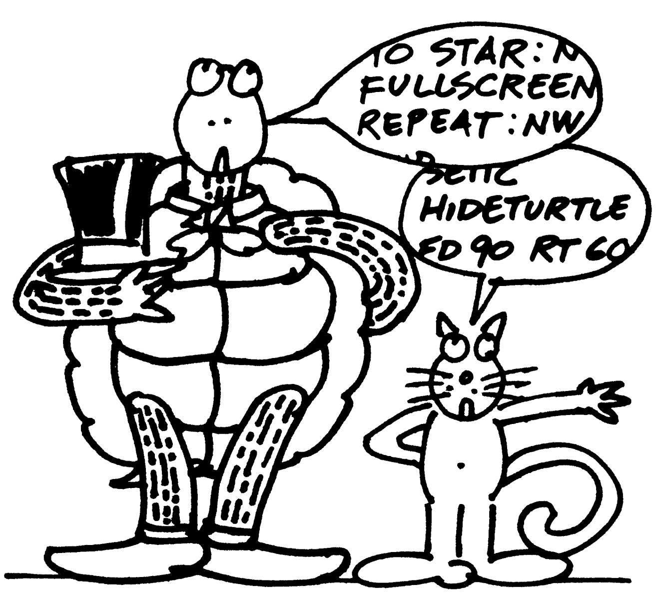
Learning Logo and seeing what someone else can do with it is fun, but the real enjoyment of Logo is in actually using it to bring an idea of your own to life on your computer's display screen. You've got all the tools you need to do this now, so pick a project or activity that interests you and enjoy your Commodore 64 Logo.

New Commands Introduced In This Chapter

Following is a list of the procedures contained in the MUSIC file on the Utilities Disk. They are available for use once you have read this file into your workspace.

Procedure
SSH
SSHER
TEMPO
PLAY
WAVE
ATTACK
DECAY
SUSTAIN
RELEASE
PULSE

How to Create Your Own Logo Activity Disk



The Logo Activity Time sections in this book require a Logo Activity Disk. Even if you've never prepared such a disk, you can easily make your own by following the step-by-step instructions given here.

Getting Ready

To begin, you'll need a blank disk that you have prepared, or initialized, for your Commodore 64 computer. You can purchase a box of blank disks at any computer store, or you can re-initialize a disk on which you've already stored information. Since initializing a disk erases all information on it, be careful *not* to use a disk that contains programs or information you want to save.

The procedure for initializing a disk on the Commodore 64 is described in detail under the heading Initializing a Blank Disk in Chapter 6. If you need help, follow the instructions in that section. After you've initialized the disk, use a felt-tipped pen to write "Logo Activity Disk" either on a new label or, if you're reusing a disk, on the label that's already on it.

Now, load Logo into the computer, using the Logo language disk that came with your Commodore 64 Logo package. Instructions for doing this are provided in the section titled Starting Up the System in Chapter 2. After Logo is loaded, take the Logo language disk out of the disk drive and replace it with your newly initialized disk.

Entering the Procedures

Now, just follow the instructions given here, one step at a time, to enter the chapter-by-chapter procedure listings in this appendix. If you have any trouble understanding the instructions or entering the procedures, refer to the keyboard sections of Chapter 2, the section titled Writing a Procedure in Chapter 6, and the section titled Modifying Procedures in the Edit Mode in Chapter 7.

For each procedure, you will need to do the following:

1. Type GOODBYE and press <RETURN> to clear the computer's workspace of any previous material.

2. Type EDIT and press <RETURN> to put the display screen in the Edit mode.

3. Type all the procedures for each activity exactly as they are listed. Remember to pay attention to spacing.

4. Hold the <CTRL> key down and press C to exit the Edit mode.

5. When the cursor reappears on your screen, transfer the procedures from your computer's memory to your Activity Disk as a file. Do this by typing SAVE, a space, a quotation mark, and the name of the activity. Then, press

<RETURN>. For example, you transfer Chapter 3's MAZE activity to the Activity Disk by typing:

```
SAVE "MAZE
```

---

6. When you've finished, take the Activity Disk out of the disk drive and put it away in a safe place until you're ready to use it for Logo Activity Time.

If you want to stop and turn off the computer before you've entered the procedures for all the activities, finish typing the one you're working on, press <CTRL> C, save it on the initialized disk, and then turn off the computer. If you aren't able to finish typing a procedure before you turn off the computer, you'll need to go back to the point where you last saved on the disk to resume typing—everything you typed from that point to where you were when you turned off the computer will be lost.

To make the procedures more meaningful for you, a brief explanation of what each procedure does is included with each chapter listing. These explanations will help you better understand how procedures are written to accomplish a specific task and how they can be linked together to produce a more general result. Later on, when you have mastered some Logo techniques and want to try modifying these activities, you'll also find them helpful in locating procedures that do specific things.

Chapter 3: MAZE

MAZE calls the SCREEN subprocedure (which sets up the screen), sets the variables "COUNT (the number of moves taken) and "HITS (the number of times the turtle hits one of the walls of the maze) to zero, and calls the BEGIN subprocedure.

BEGIN calls the ASK.DIRECTION, ASK.DISTANCE, and CHANGE.-DISTANCE subprocedures, increases the count of the number of moves by one each time, and calls itself, returning to its own beginning and starting again (recursion).

ASK.DIRECTION asks which direction the turtle should face.

ASK.DISTANCE asks how many steps the turtle should take.

CHECK.NUMBER checks the number requested in ASK.DISTANCE to make sure it's a positive whole number.

CHANGE.DISTANCE uses the number requested in ASK.DISTANCE as the variable "CHTR and moves the turtle that number of steps. Then it uses BORDER to see if a move results in the turtle hitting a wall of the maze. If it does, the procedure puts the turtle back 10 steps and increases the number of hits by one. BORDER and CHECK use these three commands: LOCAL, .EXAMINE, and BITAND. These three commands aren't covered in this book because together they deal with how the

Commodore 64 stores the screen in memory and aren't crucial to understanding and using Logo. They're used here simply as a shortcut.

RESULT prints the result of the activity after the turtle reaches the top of the maze.

```
TO MAZE
 HT SCREEN
 PU SETXY 0 (-55) SETH 0 ST
 MAKE "COUNT 0 MAKE "HITS 0
 BEGIN
END

TO SCREEN
 SINGLECOLOR DRAW BG 6 PC 7 PU
 SETXY (-15) (-55) PD
 SETX (-150) SETY 125 SETX (-15)
 PU SETX 15 PD
 SETX 150 SETY (-55) SETX 15
 PU SETXY 90 5 PD
 SETY 35 SETX 120 SETY (-25)
 SETX (-90) SETY 5 SETX (-65)
 PU SETXY (-35) (-25) PD
 SETY 5 SETX 60 SETX 15 SETY 35 SETX 60
 SETY 65 SETX 90 SETY 95 SETX 120
 PU SETXY 150 65 PD SETX 120 PU
 SETX 60 PD SETX 30 SETY 95 SETX (-30)
 PU SETXY 60 125 PD SETY 95 PU
 SETXY (-60) 125 PD SETY 95 PU
 SETXY (-150) (-25) PD SETX (-120)
 SETY 35 PU SETXY (-90) 95 PD
 SETX (-120) SETY 65 PD SETX (-90)
 SETY 35 SETX (-15) SETY 65
 SETX 0 SETX (-45)
END

TO BEGIN
 ASK.DIRECTION ASK.DISTANCE
 MAKE "STEPS 0
 CHANGE.DISTANCE :CHTR MAKE "COUNT :COUNT+1
 BEGIN
END

TO ASK.DIRECTION
 PRINT [WHAT DIRECTION DO YOU WANT TO FACE?]
 PRINT [UP=U; DOWN=D; LEFT=L; RIGHT=R]
 MAKE "CHTR FIRST REQUEST
```

```
  IF :CHTR="U THEN SETH 0 STOP
  IF :CHTR="D THEN SETH 180 STOP
  IF :CHTR="L THEN SETH 270 STOP
  IF :CHTR="R THEN SETH 90 STOP
  CLEARTEXT PRINT [PLEASE SELECT ONLY U, D, L OR R.]
  ASK.DIRECTION
END

TO ASK.DISTANCE
  PRINT [HOW FAR DO YOU WANT TO MOVE?]
  MAKE "CHTR FIRST REQUEST CHECK.NUMBER
END

TO CHECK.NUMBER
  TEST NUMBER? :CHTR
  IFFALSE THEN CLEARTEXT PRINT [PLEASE GIVE A POSITIVE INTEGER ANSWER.]
    ASK.DISTANCE STOP
END

TO CHANGE.DISTANCE :CHTR
  IF :STEPS > :CHTR THEN CLEARTEXT STOP
  FD 3 MAKE "STEPS :STEPS+3
  IF BORDER THEN BK 10 MAKE "HITS :HITS+1 CLEARTEXT STOP
  IF YCOR > 125 THEN RESULT TOPLEVEL
  CHANGE.DISTANCE :CHTR
END

TO BORDER
  LOCAL "T MAKE "T .EXAMINE 53279
  OP 0 < BITAND .EXAMINE 53279 CHECK WHO
END

TO CHECK :K
  OP ITEM 1+:K [1 2 4 8 16 32 64 128]
END

TO RESULT
  TEXTSCREEN CLEARTEXT MAKE "COUNT :COUNT+1
  PRINT []
  PRINT [CONGRATULATIONS! YOU MADE IT ALL THE]
  (PRINT [WAY THROUGH THE MAZE IN ONLY] :COUNT )
  PRINT [MOVES. AND, YOU ONLY HIT THE WALL]
  (PRINT :HITS [TIMES. I'LL BET YOU CAN DO IT] )
  PRINT [IN EVEN FEWER MOVES, AND WITH FEWER]
  PRINT [HITS, NEXT TIME.]
  PRINT []
```

```
 PRINT [IF YOU WANT TO TRY AGAIN, JUST TYPE:]
 PRINT [MAZE <RETURN>.]
END
```

Chapter 5: FIND.ME

FIND.ME calls the SETSCREEN and HIDE subprocedures, sets the variable "COUNT (the number of guesses taken) to zero, and calls the START.GAME subprocedure.

SETSCREEN draws the horizontal and vertical (X and Y) coordinate lines on the screen.

HIDE sets the variable "A (the number of positions to which the turtle moves before coming to rest) to zero and calls the DASH.AROUND subprocedure.

DASH.AROUND selects an X coordinate and a Y coordinate randomly, moves the turtle to that position, and calls itself. The procedure stops when the turtle reaches the tenth position.

START.GAME increases the number of guesses ("COUNT) by one and calls the ASK.FOR.GUESS subprocedure. It then checks to see if both guesses are correct. If they are, it draws a STAR, goes to the GIVE.RESULT subprocedure and stops. Otherwise, it calls MAKE.RECTANGLE to draw a rectangle. It then checks to see how many guesses have been made. If four guesses have been made, it calls the GIVE.RESULT subprocedure and stops. Otherwise, it calls itself and begins again.

ASK.FOR.GUESS clears the screen of text, calls the ASK.FOR.XGUESS subprocedure, lets the variable "P represent the X guess, and calls CHANGEX. It then clears the text, calls ASK.FOR.YGUESS, and "P represent the Y guess, and calls the CHANGEY subprocedure.

ASK.FOR.XGUESS checks to see if the X coordinate of the turtle has been correctly guessed yet. If it has, the procedure ends. If it hasn't, it checks to see if the Y coordinate of the turtle has been correctly guessed yet. If it has, it displays that information on the screen. It then asks for a new X guess and sets "XGUESS equal to the number typed in. It then makes sure the guess is a whole number and that it is within the interval of values that were specified. If any of these tests fail, it asks for another guess. If all the tests pass, the procedure ends. ASK.FOR.YGUESS does the same as ASK.FOR.XGUESS for the Y coordinate.

GIVE.RESULT prints the correct X and Y values, the best X and Y guesses, the error score for X and Y individually, and the total error score for X and Y together (the sum of the X error and Y error).

CHANGEX changes the values of the variables that represent the corners of the rectangle drawn in MAKE.RECTANGLE so that the last X guess is one of the corners. CHANGEY does the same for the last Y guess.

```
TO FIND.ME
 SETSCREEN HIDE
 MAKE "Q (-155)
 MAKE "R 155
 MAKE "S (-70)
 MAKE "T 130
 MAKE "COUNT 0
 START.GAME
END

TO SETSCREEN
 NODRAW DRAW TEXTCOLOR 1 BG 6 PC 7
 FD 130 BK 200 FD 70 RT 90
 FD 155 BK 310 FD 155 LT 90
END

TO HIDE
 MAKE "A 0
 DASH.AROUND SETXY (:X) (:Y)
END

TO DASH.AROUND
 PU RANDOMIZE
 MAKE "X (RANDOM 301)-150
 MAKE "Y (RANDOM 151)-50
 IF :A=10 THEN STOP
 MAKE "A :A+1
 SETXY (:X) (:Y) DASH.AROUND
END

TO START.GAME
 MAKE "COUNT :COUNT+1 ASK.FOR.GUESS
 IF ALLOF :Q=:R :S=:T THEN STAR GIVE.RESULT STOP
 MAKE.RECTANGLE
 IF :COUNT=4 THEN GIVE.RESULT STOP
 START.GAME
END

TO MAKE.RECTANGLE
 HT PU SETXY (:Q) (:S)
 PD SETY (:T) SETX (:R) SETY (:S) SETX (:Q) PU
 SETXY (:X) (:Y) ST
END

TO ASK.FOR.GUESS
 CLEARTEXT MAKE "P ASK.FOR.XGUESS
 CHANGEX
```

```
 CLEARTEXT MAKE "P ASK.FOR.YGUESS
 CHANGEY
END

TO ASK.FOR.XGUESS
 IF :Q=:R THEN OUTPUT :Q STOP
 IF :S=:T THEN PRINT (SENTENCE [Y=] :S [IS RIGHT!] )
 PRINT (SENTENCE [MY X COORDINATE IS BETWEEN] :Q [AND] :R )
 PRINT [WHAT DO YOU THINK MY X COORDINATE IS?]
 PR.GUESS
 MAKE "XGUESS FIRST REQUEST TEST :XGUESS < :Q
 IFTRUE THEN CLEARTEXT PRINT [THAT VALUE IS TOO SMALL] PRINT ASK.FOR.XGUESS
 TEST :XGUESS > :R
 IFTRUE THEN CLEARTEXT PRINT [THAT VALUE IS TOO LARGE] PRINT ASK.FOR.XGUESS
 TEST INTEGER :XGUESS=:XGUESS
 IFFALSE THEN CLEARTEXT PRINT [PLEASE USE A WHOLE NUMBER GUESS] PRINT
  ASK.FOR.XGUESS
 OUTPUT :XGUESS
END

TO ASK.FOR.YGUESS
 IF :S=:T THEN OUTPUT :S STOP
 IF :Q=:R THEN PRINT (SENTENCE [X=] :Q [IS RIGHT!] )
 PRINT (SENTENCE [MY Y COORDINATE IS BETWEEN] :S [AND] :T )
 PRINT [WHAT DO YOU THINK MY Y COORDINATE IS?]
 PR.GUESS
 MAKE "YGUESS FIRST REQUEST
 IF :YGUESS < :S THEN CLEARTEXT PRINT [THAT VALUE IS TOO SMALL] PRINT
  ASK.FOR.YGUESS
 IF :YGUESS > :T THEN CLEARTEXT PRINT [THAT VALUE IS TOO LARGE] PRINT
  ASK.FOR.YGUESS
 TEST INTEGER :YGUESS=:YGUESS
 IFFALSE THEN CLEARTEXT PRINT [PLEASE USE A WHOLE NUMBER GUESS] PRINT
  ASK.FOR.YGUESS
 OUTPUT :YGUESS
END

TO GIVE.RESULT
 NODRAW
 MAKE "XERROR SQRT (:X - :XGUESS) * (:X -:XGUESS)
 MAKE "YERROR SQRT (:Y - :YGUESS) * (:Y -:YGUESS)
 PRINT [] PRINT [] TEXTCOLOR 7
 PRINT [YOUR FINAL AND CLOSEST GUESS]
 PRINT [FOR THE TURTLE'S POSITION WAS:]
 PRINT [] PR.SPC 10 TEXTCOLOR 1
```

```
 PRINT (SENTENCE [X=] :XGUESS [AND Y=] :YGUESS )
 PRINT [] TEXTCOLOR 7
 PRINT [SINCE THE TURTLE WAS ACTUALLY AT]
 PRINT [] PR.SPC 10 TEXTCOLOR 1
 PRINT (SENTENCE [X=] :X [AND Y=] :Y )
 PRINT [] TEXTCOLOR 7
 PRINT [YOU WERE OFF BY]
 PRINT [] PR.SPC 10 TEXTCOLOR 1
 PRINT (SENTENCE :XERROR [IN THE X COORDINATE AND] )
 PR.SPC 10
 PRINT (SENTENCE :YERROR [IN THE Y COORDINATE] )
 PRINT [] TEXTCOLOR 7
 PRINT [YOUR TOTAL ERROR SCORE IS THEREFORE]
 PRINT [] PR.SPC 10 TEXTCOLOR 1
 PRINT (SENTENCE :XERROR [ + ] :YERROR [ = ] :XERROR+:YERROR )
 PRINT [] PRINT []
END

TO CHANGEX
 IF :P < :X THEN MAKE "Q :P
 IF :P > :X THEN MAKE "R :P
 IF :P=:X THEN MAKE "Q :P MAKE "R :P
END

TO CHANGEY
 IF :P < :Y THEN MAKE "S :P
 IF :P > :Y THEN MAKE "T :P
 IF :P=:Y THEN MAKE "S :P MAKE "T :P
END

TO PR.GUESS
 PRINT1 (SENTENCE 5 - :COUNT [GUESS] )
 IF :COUNT=4 THEN PRINT1 [' LEFT:'] ELSE PRINT1 [ES LEFT:]
END

TO PR.SPC :NSPC
 REPEAT :NSPC [PRINT1 [' ']]
END

TO STAR
 PC 1 PU
 BK 20 PD LT 9
 REPEAT 20 [FD 40 RT (360/20)*9]
 HT
 REPEAT 2000 []
END
```

Chapter 6: SPELL'NDRAW

The E procedure erases the screen. The L and R procedures turn the turtle counterclockwise and clockwise respectively. B and F move the turtle backward and forward with its pen up. P, O,H,C,T, and S draw the geometric shapes described in Chapter 6. TAB, CR, TC, and CPRINT aid in formatting the screen. ? displays the names of the commands in SPELL 'NDRAW.

```
TO E
 FULLSCREEN BG 6 PC 7 DRAW
END

TO L
 FULLSCREEN BG 6 PC 7 LT 15
END

TO R
 FULLSCREEN BG 6 PC 7 RT 15
END

TO B
 FULLSCREEN BG 6 PC 7 PU BK 10 PD
END

TO F
 FULLSCREEN BG 6 PC 7 PU FD 10 PD
END

TO P
 FULLSCREEN BG 6 PC 7 RT 45
 REPEAT 2 [FD 50 RT 45 FD 100 RT 135] LT 45
END

TO O
 FULLSCREEN BG 6 PC 7 LT 90 PU FD 15 RT 45 PD
 REPEAT 8 [FD 30 RT 45]
 RT 135 PU FD 15 LT 90 PD
END

TO H
 FULLSCREEN BG 6 PC 7 LT 60
 REPEAT 6 [FD 40 RT 60] RT 60
END

TO C
 FULLSCREEN BG 6 PC 7
 REPEAT 36 [FD 6 RT 10]
END
```

```
TO T
 FULLSCREEN BG 6 PC 7
 LT 90 PU FD 35 RT 120 PD
 REPEAT 3 [FD 70 RT 120]
 RT 60 PU FD 35 LT 90 PD
END

TO S
 FULLSCREEN BG 6 PC 7
 REPEAT 4 [FD 60 RT 90]
END

TO CPRINT :TEXT
 TC :CAPCOL PRINT1 FIRST :TEXT
 TC :LOWCOL PRINT1 BUTFIRST :TEXT
END

TO TAB :N
 CURSOR :N ITEM 2 CURSORPOS
END

TO TC :COL
 TEXTCOLOR :COL
END

TO ?
 TEXTSCREEN TEXTBG 4 CLEARTEXT
 MAKE "CAPCOL 1
 MAKE "LOWCOL 7
 MAKE "C1 8 MAKE "C3 24
 TC :LOWCOL
 PRINT [HERE ARE THE COMMANDS IN SPELL'NDRAW...]
 CR
 TAB :C1 CPRINT "ERASE
 PRINT " ' SCREEN'
 CR CR
 TC :LOWCOL PRINT [DIRECTIONS:]
 CR
 TAB :C1 CPRINT "RIGHT
 TAB :C3 CPRINT "LEFT
 CR
 TAB :C1 CPRINT "FORWARD
 TAB :C3 CPRINT "BACKWARD
 CR CR CR
 TC :LOWCOL PRINT [SHAPES:] CR
 TAB :C1 CPRINT "TRIANGLE
 TAB :C3 CPRINT "SQUARE
```

```
 CR
 TAB :C1 CPRINT" PARALLELOGRAM
 TAB :C3 CPRINT "HEXAGON
 CR
 TAB :C1 CPRINT "OCTAGON
 TAB :C3 CPRINT "CIRCLE
 CR CR CR
 TC :LOWCOL PRINT [YOU CAN ALSO TYPE NORMAL LOGO COMMANDS]
 PRINT [IF YOU WANT...]
 CR
 TC :CAPCOL
END

TO CR
 PRINT [ ]
END
```

Chapter 7: TIC.TAC

TIC.TAC sets the variables "USED (cells that have marks in them), "H (cells that have the player's mark in them) and "C (cells that have the computer's mark in them), then calls the BOARD and ASK.FIRST subprocedures.

BOARD draws a tic-tac-toe grid on the screen and calls the subprocedures ONE through NINE, which draw those numbers on the board.

ASK.FIRST asks whether the player wants to move first. If the answer is yes, it calls the ASK.MOVE subprocedure. Otherwise, it calls the MAKE. MOVE subprocedure.

ASK.MOVE asks the player in which cell to put a square. It uses the MEMBER? command to check whether the cell selected is already filled. If it is, it asks that another selection be made. ASK.MOVE then calls the HMOVES, POSITION, CHECK.H, and MAKE.MOVE subprocedures.

MAKE.MOVE randomly selects one of the unfilled cells and calls the CMOVES, USED, POSITION, CHECK.C, and ASK.MOVE subprocedures.

POSITION positions the turtle at the cell selected by the player or the computer for the next mark.

SQUARE draws a square.

TRIANGLE draws a triangle.

USED adds the cell selected by the player or the computer to the list of filled-in cells (:USED).

HMOVES adds the cell selected by the player to the list of cells with the player's mark in them (:H).

CMOVES adds the cell selected by the computer to the list of cells with the computer's mark in them (:C).

CHECK.H analyzes the cells with the player's mark in them to determine whether the player has won. If the player has won, it calls the HWINS subprocedure and stops. Otherwise, it checks to see if all the cells are filled. If so, it calls the NO.WIN subprocedure.

CHECK.C is similar to CHECK.H, but analyzes whether the computer has won. If it has, CHECK.C calls the CWINS subprocedure.

HWINS displays a message saying that the player won.

CWINS displays a message saying that the computer won.

NO.WIN displays a message saying that the game is a draw.

```
TO TIC.TAC
 DRAW SPLITSCREEN CLEARTEXT
 MAKE "USED [ ]
 MAKE "H [ ]
 MAKE "C [ ]
 MAKE "NUM 0
 HT BG 6 BOARD
 ASK.FIRST
END

TO BOARD
 PC 7
 PU SETXY (-75) 50 SETH 90 PD FD 150
 PU SETXY (-75) 0 SETH 90 PD FD 150
 PU SETXY (-25) (-50) SETH 0 PD FD 150
 PU SETXY 25 (-50) SETH 0 PD FD 150
 PU SETXY (-50) 55 SETH 0 PD ONE
 PU SETXY 0 55 SETH 0 PD TWO
 PU SETXY 50 53 SETH 0 PD THREE
 PU SETXY (-50) 3 SETH 0 PD FOUR
 PU SETXY 0 5 SETH 0 PD FIVE
 PU SETXY 50 5 SETH 0 PD SIX
 PU SETXY (-50) (-45) SETH 0 PD SEVEN
 PU SETXY 0 (-45) SETH 0 PD EIGHT
 PU SETXY 50 (-45) SETH 0 PD NINE
 PU HOME PD
END

TO ASK.FIRST
 PRINT [DO YOU WANT TO GO FIRST?]
 PRINT [(TYPE YES OR NO)]
 MAKE "FIRST FIRST REQUEST
 IF :FIRST="YES THEN CLEARTEXT ASK.MOVE STOP
```

```
  IF :FIRST="NO THEN MAKE.MOVE STOP
  CLEARTEXT PRINT [PLEASE ANSWER ONLY YES OR NO]
  ASK.FIRST
END

TO ASK.MOVE
  PRINT [WHERE DO YOU WANT TO PUT YOUR SQUARE?]
  PRINT [(TYPE 1, 2, 3, 4, 5, 6, 7, 8, OR 9)]
  MAKE "CELL FIRST REQUEST
  IF MEMBER? :CELL :USED THEN CLEARTEXT PRINT [THAT POSITION IS ALREADY FILLED]
   ASK.MOVE
  IF MEMBER? :CELL [1 2 3 4 5 6 7 8 9] THEN HMOVES
   :CELL USED :CELL POSITION :CELL SQUARE CHECK.H MAKE.MOVE STOP
  CLEARTEXT
  PRINT [TYPE ONLY 1, 2, 3, 4, 5, 6, 7, 8, OR 9]
  ASK.MOVE
END

TO MAKE.MOVE
  MAKE "CELL (RANDOM 9)+1
  IF MEMBER? :CELL :USED THEN MAKE.MOVE
  CMOVES :CELL
  CLEARTEXT USED :CELL
  PRINT [I THINK I'LL PUT MY TRIANGLE IN]
  (PRINT [POSITION] :CELL )
  POSITION :CELL TRIANGLE CHECK.C
  CLEARTEXT ASK.MOVE
END

TO POSITION :CELL
  PU SETH 0
  IF :CELL=1 THEN SETXY (-60) 72
  IF :CELL=2 THEN SETXY (-10) 72
  IF :CELL=3 THEN SETXY 40 72
  IF :CELL=4 THEN SETXY (-60) 22
  IF :CELL=5 THEN SETXY (-10) 22
  IF :CELL=6 THEN SETXY 40 22
  IF :CELL=7 THEN SETXY (-60) (-28)
  IF :CELL=8 THEN SETXY (-10) (-28)
  IF :CELL=9 THEN SETXY 40 (-28)
  PD
END

TO SQUARE
  PC 3 MAKE "S 20
```

```
  REPEAT 5 [REPEAT 4 [FD :S RT 90] PU FD 0.5 RT 90 FD 0.5 LT 90 PD MAKE "S :S-1]
  MAKE "NUM :NUM+1
END

TO TRIANGLE
  SETH 30 PC 1 PD MAKE "S 20
  REPEAT 10 [REPEAT 3 [FD :S RT 120] FD 0.25 RT 90 FD 0.2 LT 90 MAKE "S :S-0.5]
  PU SETH 0 MAKE "NUM :NUM+1
END

TO USED :CELL
  MAKE "USED SENTENCE :USED :CELL
END

TO HMOVES :CELL
  MAKE "H SENTENCE :H :CELL
END

TO CMOVES :CELL
  MAKE "C SENTENCE :C :CELL
END

TO CHECK.H
  IF (ALLOF MEMBER? 1 :H MEMBER? 2 :H MEMBER? 3 :H ) THEN HWINS STOP
  IF (ALLOF MEMBER? 1 :H MEMBER? 4 :H MEMBER? 7 :H ) THEN HWINS STOP
  IF (ALLOF MEMBER? 1 :H MEMBER? 5 :H MEMBER? 9 :H ) THEN HWINS STOP
  IF (ALLOF MEMBER? 2 :H MEMBER? 5 :H MEMBER? 8 :H ) THEN HWINS STOP
  IF (ALLOF MEMBER? 3 :H MEMBER? 5 :H MEMBER? 7 :H ) THEN HWINS STOP
  IF (ALLOF MEMBER? 3 :H MEMBER? 6 :H MEMBER? 9 :H ) THEN HWINS STOP
  IF (ALLOF MEMBER? 4 :H MEMBER? 5 :H MEMBER? 6 :H ) THEN HWINS STOP
  IF (ALLOF MEMBER? 7 :H MEMBER? 8 :H MEMBER? 9 :H ) THEN HWINS STOP
  IF :NUM=9 THEN NO.WIN STOP
END

TO CHECK.C
  IF (ALLOF MEMBER? 1 :C MEMBER? 2 :C MEMBER? 3 :C ) THEN CWINS STOP
  IF (ALLOF MEMBER? 1 :C MEMBER? 4 :C MEMBER? 7 :C ) THEN CWINS STOP
  IF (ALLOF MEMBER? 1 :C MEMBER? 5 :C MEMBER? 9 :C ) THEN CWINS STOP
  IF (ALLOF MEMBER? 2 :C MEMBER? 5 :C MEMBER? 8 :C ) THEN CWINS STOP
  IF (ALLOF MEMBER? 3 :C MEMBER? 5 :C MEMBER? 7 :C ) THEN CWINS STOP
  IF (ALLOF MEMBER? 3 :C MEMBER? 6 :C MEMBER? 9 :C ) THEN CWINS STOP
  IF (ALLOF MEMBER? 4 :C MEMBER? 5 :C MEMBER? 6 :C ) THEN CWINS STOP
  IF (ALLOF MEMBER? 7 :C MEMBER? 8 :C MEMBER? 9 :C ) THEN CWINS STOP
  IF :NUM=9 THEN NO.WIN STOP
END
```

```
TO HWINS
 CLEARTEXT PRINT [CONGRATULATIONS! YOU BEAT ME.]
 PRINT [IF YOU WANT TO PLAY AGAIN, TYPE TIC.TAC] TOPLEVEL
END

TO CWINS
 CLEARTEXT PRINT [SORRY, BUT I WIN.]
 PRINT [IF YOU WANT TO PLAY AGAIN, TYPE TIC.TAC] TOPLEVEL
END

TO NO.WIN
 CLEARTEXT PRINT [THIS GAME IS A DRAW, SO NEITHER]
 PRINT [OF US WINS.]
 PRINT [IF YOU WANT TO PLAY AGAIN, TYPE TIC.TAC]
 TOPLEVEL
END

TO ONE
 FD 8
END

TO TWO
 LT 90 BK 2.5 FD 5 RT 150 FD 6
 LT 45 FD 3 LT 90 FD 5 LT 90 FD 3
END

TO THREE
 PU FD 10 PD RT 90 BK 3 FD 6
 RT 135 FD 8 LT 135 FD 5
 RT 90 FD 4 RT 90 FD 5
END

TO FOUR
 FD 10 BK 6 LT 90
 BK 3 FD 6 RT 90 FD 6
END

TO FIVE
 RT 90 BK 3 FD 6 LT 90
 FD 4 LT 90 FD 6 RT 90
 FD 4 RT 90 FD 6
END

TO SIX
 PU FD 8 LT 90 BK 3 PD
 FD 6 LT 90 FD 8 LT 90 FD 6 LT 90
 FD 4 LT 90 FD 6
END
```

```
TO SEVEN
 PU FD 8
 LT 90 PD FD 3 BK 6 LT 60 FD 8
END

TO EIGHT
 LT 90 BK 2.5 FD 5 RT 90 FD 8
 RT 90 FD 5 RT 90 FD 8 BK 4 RT 90 FD 5
END

TO NINE
 RT 90 BK 2.5 FD 5 LT 90 FD 8
 LT 90 FD 5 LT 90 FD 4 LT 90 FD 5
END
```

Chapter 9: FRACTIONS

FRACTIONS asks the player to type in four numerical values and represents these values by the names "A, "B, "C, and "D. It then calls the DRAW.FIGURES and RESULT subprocedures. DRAW.FIGURES tests the values of "A and "B to determine which subprocedure, DRAW.1.1 or DRAW.1.2, to call to draw the corresponding figure on the screen. Similarly, it tests the values of "C and "D to determine which subprocedure, DRAW.2.1 or DRAW.2.2, to call to draw the corresponding figure on the screen. The difference between DRAW.1.1 and DRAW.2.1 is that different values of "A and "B cause FIGURE to draw figures either to the left or the right of where the turtle begins drawing. The two versions of DRAW help keep the figures centered on the screen.

DRAW.1.1 and DRAW.1.2 both call FIGURE with the values of "A and "B to draw the corresponding figure on the left side of the screen. DRAW.2.1 and DRAW.2.2 do the same thing as DRAW.1.1 and DRAW.1.2 for the second figure, using the values of "C and "D.

FIGURE takes two inputs and draws the figure specified by these inputs. The figure will generally be either a star or a polygon.

RESULT checks whether the values of "A, "B, "C, and "D give fractions A/B and C/D that are equal to each other. It then prints the result of this check on the screen and asks if the player wants to play the game again. If the player does, it calls itself and starts over. If not, it clears the screen and stops.

```
TO FRACTIONS
 SPLITSCREEN
 DRAW BG 6 PC 7 CLEARTEXT
 PRINT [FOR THE FIRST FRACTION, A/B,]
 PRINT [WHAT VALUE DO YOU WANT FOR B?]
 MAKE "B FIRST REQUEST
 PRINT (SENTENCE [WHAT VALUE (BETWEEN 1 AND] :B-1 [) IS A?] )
 MAKE "A FIRST REQUEST
 IF ANYOF :A < 1 :A > (:B-1 ) THEN FRACTIONS
 CLEARTEXT
 PRINT [FOR THE SECOND FRACTION, C / D,]
 PRINT [WHAT VALUE DO YOU WANT FOR D?]
 MAKE "D FIRST REQUEST
 PRINT (SENTENCE [WHAT VALUE (BETWEEN 1 AND] :D - 1 [) IS C?] )
 MAKE "C FIRST REQUEST
 IF ANYOF :C < 1 :C > (:D-1 ) THEN FRACTIONS
 DRAW.FIGURES RESULT WRAP
END

TO RESULT
 CLEARTEXT TEST :A * :D = :B * :C
 IFTRUE THEN PRINT [THESE TWO FIGURES ARE THE SAME.]
 IFTRUE THEN PRINT (SENTENCE [THE FRACTIONS] :A [ / ] :B [AND] :C [ / ] :D )
 IFTRUE THEN PRINT [ARE EQUIVALENT.]
 IFFALSE THEN PRINT [THESE TWO FIGURES ARE NOT THE SAME.]
 IFFALSE THEN PRINT (SENTENCE [THE FRACTIONS] :A [ / ] :B [AND] :C [ / ] :D )
 IFFALSE THEN PRINT [ARE NOT EQUIVALENT.]
 PRINT [IF YOU WANT TO STOP, TYPE "NO"]
 IF RC = "NO THEN TEXTSCREEN CLEARTEXT STOP
 FRACTIONS
 DRAW CLEARTEXT BG 11 PC 1
END

TO DRAW.FIGURES
 BG 6 PC 7 ST
 TEST :A / :B < 0.5
 IFTRUE THEN DRAW.1.1 :A :B
 IFFALSE THEN DRAW.1.2 :A :B
 TEST :C / :D < 0.5
 IFTRUE THEN DRAW.2.1 :C :D
 IFFALSE THEN DRAW.2.2 :C :D
END
```

```
TO DRAW.1.1 :M :N
 PU SETXY (-120) 0 PD FIGURE :M :N
END

TO DRAW.1.2 :M :N
 PU SETXY (-40) 0 PD FIGURE :M :N
END

TO DRAW.2.1 :M :N
 PU SETXY 40 0 PD FIGURE :M :N
END

TO DRAW.2.2 :M :N
 PU SETXY 120 0 PD FIGURE :M :N
END

TO FIGURE :MULTIPLIER :NUMBER
 MAKE "ANG ( 360/:NUMBER ) * :MULTIPLIER
 REPEAT :NUMBER [FD 50 RT :ANG]
END
```

Chapter 10: SPACETURTLE

SPACETURTLE calls the SCREEN subprocedure, sets the variables "COUNT (the number of asteroids drawn) and "SHOTS (the number of shots taken) to zero, and calls the DOTS and MOVE subprocedures.

SCREEN draws a rectangular boundary to serve as the sector of space inside which the turtle moves.

DOTS calls the DOT subprocedure, increases the number of asteroids drawn (:COUNT) by one, and calls itself. The procedure ends when the number of asteroids is 26. DOT draws an asteroid.

MOVE calls the BOUNDARY? subprocedure to test whether the turtle has hit the rectangle serving as the boundary of the activity. If it has, the turtle reverses direction. MOVE then uses the RC? command to check whether a key has been pressed. If one has, it calls the CHANGE subprocedure and uses the RC command to give CHANGE the key pressed as an input value. MOVE then moves the turtle forward four steps and calls itself to begin again.

BOUNDARY? checks the X and Y coordinates of the turtle to test whether the turtle has reached the rectangular boundary. It then uses the OUTPUT command to send the result of this test to the MOVE procedure.

CHANGE uses the key pressed as its input value (:CHTR) to change the turtle's direction, to shoot the turtle's phaser, or to call the RESULT subprocedure and end the

activity. If the player shoots the phaser by pressing S, CHANGE increases the number of shots ("SHOTS) by one.

RESULT sets the screen to all text, clears the screen, and displays a message telling how many shots were fired.

```
TO SPACETURTLE
 TELL 0 SCREEN MAKE "COUNT 0
 DOTS PU HOME SETHEADING 0 PD PC 7 ST
 TELL 1 PU HOME SETHEADING 0 PD PC -1
 TELL 0 MAKE "SHOTS 0
 MOVE
END

TO SCREEN
 DOUBLECOLOR FULLSCREEN BG 6 PC 3 HT
 MAKE "X 120 MAKE "Y 96
 REPEAT 7 [PU SETXY (-:X) (-:Y) PD SETY :Y SETX :X SETY (-:Y) SETX (-:X)
  MAKE "X :X-1 MAKE "Y :Y-1]
 PC 7
END

TO DOTS
 RANDOMIZE
 MAKE "X (RANDOM 171) - 85
 MAKE "Y (RANDOM 131) - 65
 PU SETXY (:X) (:Y) PD DOT
 MAKE "COUNT :COUNT+1
 IF :COUNT= 26 THEN STOP
 DOTS
END

TO DOT
 REPEAT 4 [FD 1 RT 90]
END

TO MOVE
 TEST BOUNDARY?
 IFFALSE THEN RT 180 TELL 1 RT 180 TELL 0
 IF RC? THEN CHANGE RC
 FD 4 TELL 1 FD 4
 TELL 0 MOVE
END

TO BOUNDARY?
 IF (ANYOF XCOR < (-97) XCOR > 97 YCOR < (-75) YCOR > 75 ) THEN OUTPUT "FALSE
 OUTPUT "TRUE
END
```

```
TO CHANGE :CHTR
  IF :CHTR = "U THEN SETH 0 TELL 1 SETH 0 TELL 0 STOP
  IF :CHTR = "D THEN SETH 180 TELL 1 SETH 180 TELL 0 STOP
  IF :CHTR = "R THEN SETH 90 TELL 1 SETH 90 TELL 0 STOP
  IF :CHTR = "L THEN SETH 270 TELL 1 SETH 270 TELL 0 STOP
  IF :CHTR = "E THEN TELL 0 RESULT TOPLEVEL STOP
  IF :CHTR = "S THEN TELL 1 PC 0 PU FD 5 PD LT 45 REPEAT 30 [FD 20 BK 20 RT 3] PC -1
    LT 90 REPEAT 30 [FD 20 BK 20 RT 3] LT 45 PU BK 5 PD TELL 0 MAKE "SHOTS :SHOTS + 1
    STOP
END

TO RESULT
  CLEARTEXT TELL 1 PC 1 PU HOME SETH 0 PD
  TELL 0 SINGLECOLOR NODRAW
  PRINT [ ] PRINT [ ]
  PRINT (SENTENCE [SO FAR YOU'VE FIRED] :SHOTS [SHOTS.] )
  PRINT [TRY USING FEWER SHOTS NEXT TIME.]
END
```

Chapter 11: RELAY

RELAY sets the variables "S.1, "S.2, "S.3, and "S.4 (representing the shapes of the four sprites that will be racing) to 7, and calls the ERASE.SPRITES, BORDER, SET.1, SET.2, SET.3, SET.4, and GO.RACE subprocedures.

BORDER draws a rectangle around the outside of the screen to serve as the borders of the race track.

SET.1, SET.2, SET.3, and SET.4 set the shape, color, and size of the four sprites that race across the screen.

GO.RACE sets the four sprites in motion, one at a time, using the RANDOM 21 command to randomly move each sprite a distance between 0 and 20 steps. GO.RACE also calls the CHECK.1, CHECK.2, CHECK.3, and CHECK.4 subprocedures to determine whether a sprite has reached the far left or right border of the racing area.

CHECK.1, CHECK.2, CHECK.3, and CHECK.4 test whether sprite 1, 2, 3, or 4 has reached the left or right border. If a sprite has reached the left or right border, its shape number is reduced by one and its movement is reversed (turned 180 degrees to the right). If a sprite's shape number is zero, the procedure calls the TERM subprocedure to end the activity.

TERM clears the screen, calls the ERASE.SPRITES subprocedure, sets all four sprites back to their original shapes and sizes, and uses the TOPLEVEL command to return control to the person at the keyboard.

ERASE.SPRITES sends all the sprites back to the center of the screen, hides them, and sets the screen to the SplitScreen format, with the default background

color. It also makes the turtle the current sprite and shows the turtle at the center of the screen.

```
TO RELAY
 DRAW ERASE.SPRITES HT FULLSCREEN BG 1
 BORDER SET.1 SET.2 SET.3 SET.4
 MAKE "S.1 7 MAKE "S.2 7
 MAKE "S.3 7 MAKE "S.4 7
 GO.RACE
END

TO BORDER
 TELL 0 PU PC 4
 SETXY (-145) (-110) PD
 MAKE "X 145 MAKE "Y 110
 REPEAT 5 [SETXY (-:X) (-:Y) SETY :Y SETX :X SETY (-:Y) SETX (-:X) MAKE "X :X+1
  MAKE "Y :Y+1]
 PC 1 PU HOME
END

TO SET.1
 TELL 1 SETSHAPE 7 PC 6 BIGX BIGY
 SETXY (-110) 95 SETH 90 ST
END

TO SET.2
 TELL 2 SETSHAPE 7 PC 2 BIGX SMALLY
 SETXY (-110) 30 SETH 90 ST
END

TO SET.3
 TELL 3 SETSHAPE 7 PC 5 SMALLX BIGY
 SETXY (-97) (-10) SETH 90 ST
END

TO SET.4
 TELL 4 SETSHAPE 7 PC 7 SMALLX SMALLY
 SETXY (-97) (-80) SETH 90 ST
END

TO GO.RACE
 RANDOMIZE
 TELL 1 FD RANDOM 21 CHECK.1
 TELL 2 FD RANDOM 21 CHECK.2
 TELL 3 FD RANDOM 21 CHECK.3
 TELL 4 FD RANDOM 21 CHECK.4
 GO.RACE
END
```

```
TO CHECK.1
 IF ANYOF XCOR < (−120) XCOR > 120 THEN MAKE "S.1 :S.1−1
 IF :S.1=0 THEN TERM
 IF ANYOF XCOR < (−120) XCOR > 120 THEN SETSHAPE :S.1 RT 180 FD 20
END

TO CHECK.2
 IF ANYOF XCOR < (−120) XCOR > 120 THEN MAKE "S.2:S.2−1
 IF :S.2=0 THEN TERM
 IF ANYOF XCOR < (−120) XCOR > 120 THEN SETSHAPE :S.2 RT 180 FD 20
END

TO CHECK.3
 IF ANYOF XCOR < (−120) XCOR > 120 THEN MAKE "S.3 :S.3−1
 IF :S.3=0 THEN TERM
 IF ANYOF XCOR < (−120) XCOR > 120 THEN SETSHAPE :S.3 RT 180 FD 20
END

TO CHECK.4
 IF ANYOF XCOR < (−120) XCOR > 120 THEN MAKE "S.4 :S.4−1
 IF :S.4=0 THEN TERM
 IF ANYOF XCOR < (−120) XCOR > 120 THEN SETSHAPE :S.4 RT 180 FD 20
END

TO TERM
 DRAW ERASE.SPRITES
 TELL 1 SETSHAPE 1 SMALLX SMALLY
 TELL 2 SETSHAPE 2 SMALLX
 TELL 3 SETSHAPE 3 SMALLY
 TELL 4 SETSHAPE 4
 TOPLEVEL
END

TO ERASE.SPRITES
 MAKE "NUMBER 0
 REPEAT 8 [TELL :NUMBER PU HT HOME MAKE "NUMBER :NUMBER+1]
 TELL 0 ST PD SPLITSCREEN BG 11
END
```

After you've defined these procedures by pressing <CTRL> C, type the statement:

```
MAKE "STARTUP [RELAY]
```

and press <RETURN>. Now, save the procedures and this statement as a file by typing:

```
SAVE "RELAY
```

Chapter 12: BUILDBLOCKS

The BUILDBLOCKS activity uses a sprite shapes file that you save on the Logo Activity Disk along with the procedures given here. Before you try to create a file of your own shapes, you should first read Chapter 9, which explains the use of sprites, and Chapter 10, which gives instructions for creating your own sprite shapes and saving them as a shapes file on a disk, using the Sprite Editor (SPRED) on the Utilities Disk.

For this activity, you need to define a square as Shape 1, a triangle as Shape 2, a circle as Shape 3, a horizontal (wide) rectangle as Shape 4, and a vertical (tall) rectangle as Shape 5. Once these shapes have been defined, save them on your Activity Disk with the file name BUILDBLOCKS by typing SAVESHAPES "BUILDBLOCKS and pressing <RETURN>.

The BUILDBLOCKS procedure reads the BUILDBLOCKS shapes file from the Activity Disk into the computer and calls the SET.SPRITES and DISPLAY.SPRITES subprocedures. SET.SPRITES sets the sprites' shapes and sizes. DISPLAY.SPRITES positions the sprites on the screen. ERASE.SPRITES sends all the sprites to the center of the screen and hides them. It then makes the turtle the current sprite and shows it.

The first two letters of each of the remaining procedure names refer to one of the sprite shapes. The third letter refers to an action to be taken by the specified sprite shape. The shapes and actions are listed in Chapter 12.

```
TO BUILDBLOCKS
 DRAW SPLITSCREEN BG 0
 READSHAPES "BUILDBLOCKS
 SET.SPRITES DISPLAY.SPRITES
END

TO SET.SPRITES
 TELL 0 SETSHAPE 1 BIGX BIGY PU
 TELL 1 SETSHAPE 1 PU
 TELL 2 SETSHAPE 2 PU
 TELL 3 SETSHAPE 3 PU
 TELL 4 SETSHAPE 4 BIGX BIGY PU
 TELL 5 SETSHAPE 5 BIGX BIGY PU
 TELL 6 SETSHAPE 2 BIGX BIGY PU
 TELL 7 SETSHAPE 3 BIGX BIGY PU
END

TO DISPLAY.SPRITES
 TELL 0 SETXY (-125) 90 SETH 0 ST
 TELL 1 SETXY (-50) 67 SETH 0 ST
 TELL 2 SETXY 100 67 SETH 0 ST
 TELL 3 SETXY (-50) (-23) SETH 0 ST
```

```
  TELL 4 SETXY 25 0 SETH 0 ST
  TELL 5 SETXY 90 10 SETH 0 ST
  TELL 6 SETXY 25 90 SETH 0 ST
  TELL 7 SETXY (-125) 0 SETH 0 ST
END

TO ERASE.SPRITES
  MAKE "S 0
  REPEAT 8 [TELL :S PU HT HOME PC 1 MAKE "S :S+1]
  TELL 0 SETSHAPE 0 SMALLX SMALLY ST
END

TO BCC :S
  TELL 7 PC :S
END

TO BTC :S
  TELL 6 PC :S
END

TO VRC :S
  TELL 5 PC :S
END

TO HRC :S
  TELL 4 PC :S
END

TO SCC :S
  TELL 3 PC :S
END

TO STC :S
  TELL 2 PC :S
END

TO SSC :S
  TELL 1 PC :S
END

TO BSC :S
  TELL 0 PC :S
END

TO VRS
  TELL 5 ST
END
```

```
TO VRH
 TELL 5 HT
END

TO VRL :S
 TELL 5 SETH 270 FD :S
END

TO VRR :S
 TELL 5 SETH 90 FD :S
END

TO VRU :S
 TELL 5 SETH 0 FD :S
END

TO VRD :S
 TELL 5 SETH 180 FD :S
END

TO HRS
 TELL 4 ST
END

TO HRH
 TELL 4 HT
END

TO HRL :S
 TELL 4 SETH 270 FD :S
END

TO HRR :S
 TELL 4 SETH 90 FD :S
END

TO HRU :S
 TELL 4 SETH 0 FD :S
END

TO HRD :S
 TELL 4 SETH 180 FD :S
END

TO SCS
 TELL 3 ST
END
```

```
TO SCH
 TELL 3 HT
END

TO SCL :S
 TELL 3 SETH 270 FD :S
END

TO SCR :S
 TELL 3 SETH 90 FD :S
END

TO SCU :S
 TELL 3 SETH 0 FD :S
END

TO SCD :S
 TELL 3 SETH 180 FD :S
END

TO BCS
 TELL 7 ST
END

TO BCH
 TELL 7 HT
END

TO BCL :S
 TELL 7 SETH 270 FD :S
END

TO BCR :S
 TELL 7 SETH 90 FD :S
END

TO BCU :S
 TELL 7 SETH 0 FD :S
END

TO BCD :S
 TELL 7 SETH 180 FD :S
END

TO STS
 TELL 2 ST
END
```

```
TO STH
 TELL 2 HT
END

TO STL :S
 TELL 2 SETH 270 FD :S
END

TO STR :S
 TELL 2 SETH 90 FD :S
END

TO STU :S
 TELL 2 SETH 0 FD :S
END

TO STD :S
 TELL 2 SETH 180 FD :S
END

TO BTS
 TELL 6 ST
END

TO BTH
 TELL 6 HT
END

TO BTL :S
 TELL 6 SETH 270 FD :S
END

TO BTR :S
 TELL 6 SETH 90 FD :S
END

TO BTU :S
 TELL 6 SETH 0 FD :S
END

TO BTD :S
 TELL 6 SETH 180 FD :S
END

TO SSS
 TELL 1 ST
END
```

```
TO SSH
 TELL 1 HT
END

TO SSL :S
 TELL 1 SETH 270 FD :S
END

TO SSR :S
 TELL 1 SETH 90 FD :S
END

TO SSU :S
 TELL 1 SETH 0 FD :S
END

TO SSD :S
 TELL 1 SETH 180 FD :S
END

TO BSS
 TELL 0 ST
END

TO BSH
 TELL 0 HT
END

TO BSL :S
 TELL 0 SETH 270 FD :S
END

TO BSR :S
 TELL 0 SETH 90 FD :S
END

TO BSU :S
 TELL 0 SETH 0 FD :S
END

TO BSD :S
 TELL 0 SETH 180 FD :S
END
```

Chapter 13: TEST.ME

TEST.ME sets the screen to the TextScreen format with a clear screen. It then uses the RANDOMIZE command to make sure the numbers used in the activity are

not the same each time the activity is run. Finally, it calls the BEGIN subprocedure.

BEGIN calls the SELECT, SHOW, and CORRECT? subprocedures. It then calls itself to begin again.

SELECT randomly selects two numbers between 10 and 99 and uses the variable names "A and "B to represent them. It then represents the sum of these two numbers by the variable name "C.

SHOW prints a message on the screen asking for the sum of the two values it selected.

CORRECT? compares the value typed in to the correct sum :C. If the two values match, it calls the CORRECT subprocedure and stops. If not, it calls the INCORRECT subprocedure.

CORRECT displays a message that the answer is correct and prints a checkered line across the screen to separate the problem that was just completed from the problem that will be displayed next.

INCORRECT displays a message that the answer is incorrect, displays the correct answer, and prints a checkered line across the screen.

```
TO TEST.ME
 ND TEXTBG 11 RANDOMIZE BEGIN
END

TO BEGIN
 SELECT SHOW CORRECT?
 BEGIN
END

TO SELECT
 MAKE "A (RANDOM 90)+10
 MAKE "B (RANDOM 90)+10
 MAKE "C :A+:B
END

TO SHOW
 PRINT [WHAT IS] PRINT []
 (PRINT [' '] :A )
 (PRINT [+] :B )
 REPEAT 6 [PRINT1 [-]] PRINT []
 (PRINT1 [?] )
END

TO CORRECT?
 IF FIRST REQUEST=:C THEN CORRECT STOP
 INCORRECT
END
```

```
TO CORRECT
 PRINT [ ]
 PRINT [THAT'S RIGHT! NOW TRY ANOTHER ONE.]
 LINE TEXTCOLOR 1
END

TO INCORRECT
 PRINT [ ]
 (PRINT [SORRY. THE RIGHT ANSWER IS] :C )
 PRINT [TRY AGAIN!]
 PRINT [I'LL BET YOU GET IT RIGHT THIS TIME.]
 LINE TEXTCOLOR 1
END

TO LINE
 TEXTCOLOR RANDOM 10 PRINT [ ]
 REPEAT 38 [PRINT1 [▩]]
 PRINT [ ] PRINT [ ]
END
```

Chapter 14: BIO

BIO calls on the ASSIGN, GET.B.DATA, EXPLAIN.2, GET.DATA, MONTH.NUMS, and DAYS.BETWEEN subprocedures, sets the variable "A to the number of years between the birth date typed in and the date on which the biorhythms are to be evaluated, sets the variable "LEAP.DAYS (the number of leap days that occurred between those two dates) to zero, and calls the LEAP.DAYS, LEAP.-ENDS, BEFORE, AFTER.TOTAL, DAYS.MESSAGE, and SELECT subprocedures.

ASSIGN sets the variables "DAYS.IN (the number of days in each month of the year), "DAYS.BEFORE (the number of days in the year before the first day of each month), "DAYS.AFTER (the number of days in the year after the last day of each month), and "MONTH.LIST (the months of the year).

GET.B.DATA calls the ASK.B.MONTH, ASK.B.DAY, ASK.B.YEAR, and B.CONFIRM subprocedures. ASK.B.MONTH asks for the month, ASK.B.DAY asks for the day, and ASK.B.YEAR asks for the year of birth. B.CONFIRM asks if the birth date entered is correct.

EXPLAIN.2 displays a message asking for the date for which the biorhythms are to be evaluated.

GET.DATA calls the ASK.MONTH, ASK.DAY, ASK.YEAR, and CONFIRM subprocedures. CONFIRM asks if the date entered for evaluating the biorhythms is correct. ASK.MONTH asks for the month, ASK.DAY asks for the day, and ASK.YEAR asks for the year in which the biorhythms are to be evaluated.

MONTH.NUMS uses the MONTH subprocedure to set the variable "B.MONTH to the number corresponding to the month of birth in the month list ("MONTH.LIST). MONTH outputs the number of the month in the month list corresponding to the month specified.

DAYS.BETWEEN computes the number of days, excluding leap days, in the full years between the date of birth and the date on which the biorhythms are to be evaluated.

LEAP.DAYS computes the number of leap days that have fallen in the full years between the date of birth and the date on which the biorhythms are to be evaluated.

LEAP.ENDS checks whether a leap day fell in the year of birth and whether one has fallen in the year in which the biorhythms are to be evaluated.

BEFORE computes the number of days in the year that have elapsed up to the date on which the biorhythms are to be evaluated.

AFTER computes the number of days in the year that remain after the player's birthday.

TOTAL finds the sum of the values found in DAYS.BETWEEN, LEAP.DAYS, LEAP.ENDS, BEFORE, and AFTER.

DAYS.MESSAGE displays the number of days that have elapsed between the date of birth and the date on which the biorhythms are to be evaluated.

SELECT asks whether the player wants to look at the physical cycle, sensitivity cycle, or intellectual cycle, or end the activity. Depending on the response, SELECT calls on the PHYSICAL, SENSITIVITY, or INTELLECTUAL subprocedure, or ends the activity. PHYSICAL calls the GRAPH subprocedure to draw a physical cycle graph and provides information about the position in this cycle for the date specified. It then calls the SELECT subprocedure and ends. SENSITIVITY provides the same information about the sensitivity cycle, and INTELLECTUAL provides the same information about the intellectual cycle. GRAPH draws a sine curve to represent a physical, sensitivity, or intellectual cycle.

```
TO BIO
 ND CLEARTEXT ASSIGN GET.B.DATA
 EXPLAIN.2 GET.DATA
 MONTH.NUMS DAYS.BETWEEN
 MAKE "A :B.YEAR+1
 MAKE "LEAP.DAYS 0
 LEAP.DAYS LEAP.ENDS
 BEFORE AFTER TOTAL DAYS.MESSAGE
 SELECT
END
```

```
TO ASSIGN
 MAKE "DAYS.IN [31 28 31 30 31 30 31 31 30 31 30 31]
 MAKE "DAYS.BEFORE [0 31 59 90 120 151 181 212 243 273 304 334]
 MAKE "DAYS.AFTER [334 306 275 245 214 184 153 122 92 61 31 0]
 MAKE "MONTH.LIST [JANUARY FEBRUARY MARCH APRIL MAY JUNE JULY AUGUST
  SEPTEMBER OCTOBER NOVEMBER DECEMBER]
END

TO GET.B.DATA
 ASK.B.MONTH ASK.B.DAY ASK.B.YEAR
 B.CONFIRM
END

TO EXPLAIN.2
 CLEARTEXT
 PR [GOOD. NOW WE NEED THE DATE FOR WHICH]
 PR [YOUR BIORHYTHM VALUES ARE TO BE]
 PR [EVALUATED.] PR [ ]
END

TO GET.DATA
 ASK.MONTH ASK.DAY ASK.YEAR
 CONFIRM
END

TO MONTH.NUMS
 MAKE "K 0
 MAKE "B.MONTH.NUM MONTH :B.MONTH
 MAKE "K 0
 MAKE "MONTH.NUM MONTH :MONTH
END

TO DAYS.BETWEEN
 MAKE "A :YEAR-:B.YEAR-1
 IF :A < 0 THEN MAKE "BETWEEN 0 STOP
 MAKE "BETWEEN :A * 365
END

TO LEAP.DAYS
 IF :A=:YEAR THEN STOP
 IF INTEGER :A/4=:A/4 THEN MAKE "LEAP.DAYS :LEAP.DAYS+1
 MAKE "A :A+1
 LEAP.DAYS
END
```

```
TO LEAP.ENDS
 IF ALLOF INTEGER :B.YEAR/4=:B.YEAR/4 :B.MONTH.NUM < 3 THEN MAKE "LEAP.DAYS
   :LEAP.DAYS+1
 IF ALLOF INTEGER :YEAR/4=:YEAR/4 :MONTH.NUM > 2 THEN MAKE "LEAP.DAYS
   :LEAP.DAYS+1
END

TO BEFORE
 MAKE "BEFORE (ITEM :MONTH.NUM :DAYS.BEFORE)+:DAY
END

TO AFTER
 MAKE "AFTER (ITEM :B.MONTH.NUM :DAYS.AFTER)+(ITEM :B.MONTH.NUM :DAYS.IN)
   -:B.DAY+1
END

TO TOTAL
 MAKE "TOTAL :BETWEEN+:LEAP.DAYS+:BEFORE+:AFTER
END

TO DAYS.MESSAGE
 CLEARTEXT
 PR [BETWEEN THE DATE ON WHICH YOU WERE]
 PR [BORN AND THE DATE YOU SPECIFIED, YOU]
 PR [HAVE LIVED A TOTAL OF]
 PR :TOTAL
 PR [DAYS.] PR []
 PR [NOW LET'S LOOK AT THE VALUES OF YOUR]
 PR [THREE BIORHYTHM CYCLES ON THE DATE]
 PR [YOU SPECIFIED.] PR []
END

TO SELECT
 PR [DO YOU WANT TO:] PR []
 (PR [] [] [] [] [] [1 LOOK AT YOUR PHYSICAL CYCLE] )
 (PR [] [] [] [] [] [2 LOOK AT YOUR SENSITIVITY CYCLE] )
 (PR [] [] [] [] [] [3 LOOK AT YOUR INTELLECTUAL CYCLE] )
 (PR [] [] [] [] [] [4 END THE ACTIVITY] )
 PR [] PR [(TYPE 1, 2, 3, OR 4.)]
 MAKE "CHOICE FIRST RQ
 IF :CHOICE=1 THEN PHYSICAL CLEARTEXT SELECT STOP
 IF :CHOICE=2 THEN SENSITIVITY CLEARTEXT SELECT STOP
 IF :CHOICE=3 THEN INTELLECTUAL CLEARTEXT SELECT STOP
 IF :CHOICE=4 THEN ND CLEARTEXT PR [YOU'RE NOW OUT OF THE BIO ACTIVITY]
   TOPLEVEL
```

```
  CLEARTEXT PR [PLEASE ENTER ONLY 1, 2, 3, OR 4]
  SELECT
END

TO CONFIRM
  CLEARTEXT
  PR [THE DATE ON WHICH YOU WANT YOUR]
  PR [BIORHYTHM VALUES EVALUATED IS]
  PR [ ] (PR :MONTH :DAY [,] :YEAR ) PR [ ]
  PR [IS THAT CORRECT? (TYPE YES OR NO.)]
  MAKE "CONFIRM FIRST RQ
  IF :CONFIRM = "YES THEN PR [ ] PR [(PLEASE WAIT. I'M COMPUTING.)] STOP
  PR [ ]
  PR [IN THAT CASE, PRESS <RETURN>]
  PR [AND WE'LL START OVER AGAIN.]
  MAKE "A RQ
  GET.DATA
END

TO ASK.MONTH
  PR [ENTER THE MONTH IN WHICH YOU WANT]
  PR [YOUR BIORHYTHM VALUES TO BE]
  PR [EVALUATED. (USE THE FULL NAME]
  PR [OF THE MONTH, SUCH AS JANUARY OR]
  PR [FEBRUARY.)]
  MAKE "MONTH FIRST RQ
  IF MEMBER? :MONTH :MONTH.LIST THEN STOP
  CLEARTEXT ASK.MONTH
END

TO ASK.DAY
  CLEARTEXT
  PR [ENTER THE DAY OF THE MONTH FOR]
  PR [WHICH YOU WANT YOUR BIORHYTHM VALUES]
  PR [TO BE EVALUATED. (THIS SHOULD BE A]
  PR [POSITIVE WHOLE NUMBER BETWEEN 1]
  PR [AND 31.)]
  MAKE "DAY FIRST RQ
  IF (ALLOF NUMBER? :DAY INTEGER :DAY = :DAY :DAY > 0 :DAY < 32 ) THEN STOP
  ASK.DAY
END

TO ASK.YEAR
  CLEARTEXT
  PR [ENTER THE YEAR IN WHICH YOU WANT]
  PR [YOUR BIORHYTHM VALUES TO BE]
  PR [EVALUATED. (USE ALL FOUR DIGITS OF]
```

```
 PR [THE YEAR, SUCH AS 1950.)]
 MAKE "YEAR FIRST RQ
 IF ALLOF NUMBER? :YEAR INTEGER :YEAR=:YEAR THEN STOP
 ASK.YEAR
END

TO ASK.B.MONTH
 CLEARTEXT
 PR [IN WHAT MONTH WERE YOU BORN?]
 PR [(ENTER THE FULL NAME OF THE MONTH,]
 PR [SUCH AS JANUARY OR FEBRUARY.)]
 MAKE "B.MONTH FIRST RQ
 IF MEMBER? :B.MONTH :MONTH.LIST THEN STOP
 ASK.B.MONTH
END

TO ASK.B.DAY
 CLEARTEXT
 (PR [ON WHAT DAY IN] :B.MONTH )
 PR [WERE YOU BORN?]
 MAKE "B.DAY FIRST RQ
 IF (ALLOF NUMBER? :B.DAY INTEGER :B.DAY=:B.DAY :B.DAY > 0 :B.DAY < 32 )
  THEN STOP
 ASK.B.DAY
END

TO ASK.B.YEAR
 CLEARTEXT
 PR [IN WHAT YEAR WERE YOU BORN?]
 PR [(USE ALL FOUR DIGITS, SUCH AS 1955.)]
 MAKE "B.YEAR FIRST RQ
 IF ALLOF NUMBER? :B.YEAR INTEGER :B.YEAR=:B.YEAR THEN STOP
 ASK.B.YEAR
END

TO B.CONFIRM
 CLEARTEXT
 PR [YOU WERE BORN ON] PR []
 (PR :B.MONTH :B.DAY [,] :B.YEAR )
 PR []
 PR [IS THAT CORRECT? (TYPE YES OR NO)]
 MAKE "B.CONFIRM FIRST RQ
 IF :B.CONFIRM="YES THEN STOP
 PR []
 PR [IN THAT CASE, PRESS <RETURN> AND]
 PR [WE'LL START OVER AGAIN.]
```

```
  MAKE "A RQ
  GET.B.DATA
END

TO MONTH :A
  REPEAT 12 [MAKE "K :K+1 IF :A=(ITEM :K :MONTH.LIST) THEN MAKE "B :K]
  OUTPUT :B
END

TO GRAPH
  DRAW SPLITSCREEN BG 0 PC 1
  PU HT SETXY (-140) 37.5 SETH 90 PD
  FD 280 PU SETXY (-125) 37.5 PD
  MAKE "A (-125)
  REPEAT 50 [MAKE "A :A+5 SETXY (:A) (37.5 + 50 * (SIN 1.44 * (:A+125)))]
  PU HOME PD
END

TO PHYSICAL
  MAKE "A INTEGER (:TOTAL/23)
  MAKE "B :TOTAL-(:A*23)
  MAKE "C ((250*:B/23)-125)
  GRAPH
  PU SETXY (:C) (-27.5) SETH 0 PC 5
  REPEAT 13 [PD FD 5 PU FD 5]
  PU HOME PC 1
  (PR :MONTH :DAY [,] :YEAR [FALLS] )
  (PR :B [DAYS INTO A 23 DAY PHYSICAL CYCLE] )
  CONTIN CLEARTEXT
  PR [ON A SCALE FROM -1 TO +1,]
  PR [YOUR PHYSICAL LEVEL FOR THAT DATE]
  (PR [IS] SIN (( 360 * :B ) / 23) )
  CONTIN
  ND CLEARTEXT SELECT
END

TO SENSITIVITY
  MAKE "A INTEGER (:TOTAL/28)
  MAKE "B :TOTAL-(:A*28)
  MAKE "C ((250*:B/28)-125)
  GRAPH
  PU SETXY (:C) (-27.5) SETH 0 PC 5
  REPEAT 13 [PD FD 5 PU FD 5]
  PU HOME PC 1
  (PR :MONTH :DAY [,] :YEAR [FALLS] :B )
  PR [DAYS INTO A 28 DAY SENSITIVITY CYCLE]
  CONTIN CLEARTEXT
```

```
 PR [ON A SCALE FROM -1 TO +1,]
 PR [YOUR SENSITIVITY LEVEL FOR THAT DATE]
 (PR [IS] SIN (( 90 * :B )/7) )
 CONTIN
 ND CLEARTEXT SELECT
END

TO INTELLECTUAL
 MAKE "A INTEGER (:TOTAL/33)
 MAKE "B :TOTAL-(:A * 33)
 MAKE "C ((250 * :B/33) - 125)
 GRAPH
 PU SETXY (:C) (-27.5) SETH 0 PC 5
 REPEAT 13 [PD FD 5 PU FD 5]
 PU HOME PC 1
 (PR :MONTH :DAY [,] :YEAR [FALLS] :B )
 PR [DAYS INTO A 33 DAY INTELLECTUAL CYCLE.]
 CONTIN CLEARTEXT
 PR [ON A SCALE FROM -1 TO +1,]
 PR [YOUR INTELLECTUAL LEVEL FOR THAT DATE]
 (PR [IS] SIN (( 120 * :B )/11) )
 CONTIN
 ND CLEARTEXT SELECT
END

TO CONTIN
 PRINT1 [(PRESS <RETURN> TO CONTINUE)]
 MAKE "XX RQ
END
```

Chapter 15: MY.WORD

MY.WORD sets the screen to the TextScreen format, clears the screen, sets the variables "N (the number of the letter that will be shown as a clue) and "NUM.GUESS (the number of guesses taken) to zero, and calls the SET.UP, SET.BLANKS, SET.LETTERS, and DISPLAY subprocedures.

SET.UP sets up the variable "CHOICES as a list of words from which the computer will choose one for the player to guess. It then sets the variable "NUM to a randomly selected number that will represent the word on the list that the player is to try to guess. Finally, it sets the variable "NUM.2 to one and calls the PICK.WORD subprocedure.

PICK.WORD picks the word on the "CHOICES list corresponding to the randomly selected value of "NUM and uses the variable "SELECTION to represent that word.

SET.BLANKS sets each of the variables "1 through "8 to a dash. SET.LETTERS sets the variables "11 through "18 to the first through eighth letters of the word the player is trying to guess ("SELECTION).

DISPLAY sets the variable "SO.FAR to represent a "word" made up entirely of dashes. It calls the WHICH? subprocedure to replace one of the dashes in "SO.FAR with the corresponding letter from :SELECTION, displays the value of "SO.FAR, and asks the player to guess its word. If the guess is correct, it calls the RESULT.1 subprocedure and ends the activity. Otherwise, it increases the number of guesses taken ("NUM.GUESS) by one. If the player has used up all five guesses, DISPLAY calls the RESULT.2 subprocedure and ends the activity. Otherwise, it displays a message saying the guess is incorrect, prints a line of asterisks to separate that guess from the next guess, and calls its own name to begin again.

WHICH? sets the variable "N to an integer between 1 and 8. It then calls on the corresponding subprocedures CHANGE.1 through CHANGE.8, which replace one dash in the variable "SO.FAR with the corresponding letter in the word the player is trying to guess ("SELECTION).

RESULT.1 switches the screen to the TextScreen format, with a clear screen, and displays a message saying the guess is correct. RESULT.2 switches the screen to the TextScreen format, with a clear screen, and displays a message saying all five guesses have been used.

```
TO MY.WORD
 TEXTSCREEN CLEARTEXT RANDOMIZE
 MAKE "N 0
 MAKE "NUM.GUESS 0
 SET.UP SET.BLANKS SET.LETTERS
 DISPLAY
END

TO SET.UP
 MAKE "CHOICES [AMERICAN BALLOONS BIRTHDAY CARRIAGE COMPUTER
  DOORKNOB FOOTBALL KANGAROO SURPRISE UMBRELLA]
 MAKE "NUM (RANDOM 9)+1
 MAKE "NUM.2 1
 PICK.WORD :CHOICES
END

TO PICK.WORD :CHOICES
 IF :NUM.2=:NUM THEN MAKE "SELECTION FIRST :CHOICES STOP
 MAKE "NUM.2 :NUM.2+1
 PICK.WORD BUTFIRST :CHOICES
END
```

```
TO SET.BLANKS
 MAKE "1 "– MAKE "2 "–
 MAKE "3 "– MAKE "4 "–
 MAKE "5 "– MAKE "6 "–
 MAKE "7 "– MAKE "8 "–
END

TO SET.LETTERS
 MAKE "S :SELECTION
 MAKE "11 FIRST :S MAKE "S BUTFIRST :S
 MAKE "12 FIRST :S MAKE "S BUTFIRST :S
 MAKE "13 FIRST :S MAKE "S BUTFIRST :S
 MAKE "14 FIRST :S MAKE "S BUTFIRST :S
 MAKE "15 FIRST :S MAKE "S BUTFIRST :S
 MAKE "16 FIRST :S MAKE "S BUTFIRST :S
 MAKE "17 FIRST :S MAKE "S BUTFIRST :S
 MAKE "18 FIRST :S
END

TO DISPLAY
 WHICH?
 PRINT [WHAT DO YOU THINK MY WORD IS?]
 PRINT [ ]
 MAKE "SO.FAR (WORD :1 :2 :3 :4 :5 :6 :7 :8)
 PRINT :SO.FAR PRINT [ ]
 MAKE "GUESS FIRST REQUEST
 IF :GUESS = :SELECTION THEN RESULT.1 TOPLEVEL
 MAKE "NUM.GUESS :NUM.GUESS + 1
 IF :NUM.GUESS = 5 THEN RESULT.2 TOPLEVEL
 PRINT [SORRY, THAT'S NOT RIGHT. TRY AGAIN.]
 PRINT [ ]
 PRINT [* * * * * * * * * * * * * * * * * * * *]
 DISPLAY
END

TO WHICH?
 MAKE "N (RANDOM 7) + 1
 IF :N = 1 THEN CHANGE.1
 IF :N = 2 THEN CHANGE.2
 IF :N = 3 THEN CHANGE.3
 IF :N = 4 THEN CHANGE.4
 IF :N = 5 THEN CHANGE.5
 IF :N = 6 THEN CHANGE.6
 IF :N = 7 THEN CHANGE.7
 IF :N = 8 THEN CHANGE.8
END
```

```
TO CHANGE.1
 IF :1=:11 THEN WHICH? STOP
 MAKE "1 :11
END

TO CHANGE.2
 IF :2=:12 THEN WHICH? STOP
 MAKE "2 :12
END

TO CHANGE.3
 IF :3=:13 THEN WHICH? STOP
 MAKE "3 :13
END

TO CHANGE.4
 IF :4=:14 THEN WHICH? STOP
 MAKE "4 :14
END

TO CHANGE.5
 IF :5=:15 THEN WHICH? STOP
 MAKE "5 :15
END

TO CHANGE.6
 IF :6=:16 THEN WHICH? STOP
 MAKE "6 :16
END

TO CHANGE.7
 IF :7=:17 THEN WHICH? STOP
 MAKE "7 :17
END

TO CHANGE.8
 IF :8=:18 THEN WHICH? STOP
 MAKE "8 :18
END

TO RESULT.1
 TEXTSCREEN CLEARTEXT PRINT [ ]
 (PRINT [CONGRATULATIONS! MY WORD WAS] :SELECTION )
 PRINT [AND YOU CORRECTLY GUESSED IT. YOU'RE]
 PRINT [PRETTY GOOD AT THIS, AREN'T YOU?]
 PRINT [ ]
```

```
 PRINT [IF YOU WANT TO TRY AGAIN, JUST TYPE]
 PRINT ["MY.WORD" AND PRESS <RETURN>.]
END

TO RESULT.2
 TEXTSCREEN CLEARTEXT PRINT [ ]
 PRINT [I'M SORRY. THOSE WERE GOOD TRIES BUT]
 PRINT [YOUR 5 GUESSES ARE ALL USED UP.]
 PRINT [ ]
 PRINT [IF YOU WANT TO TRY AGAIN, TYPE]
 PRINT ["MY.WORD" AND PRESS <RETURN>.]
END
```

Chapter 16: MEMO.PAD

MEMO.PAD sets the screen to the TextScreen format, with a clear screen, and calls the ASK subprocedure. ASK asks whether the player wants to look up an event or a date and calls either the LOOK.UP.EVENT subprocedure or the LOOK.UP.DATE subprocedure.

LOOK.UP.EVENT asks the player to type in an event to be looked up. If the event is on its list of events ("EVENT.LIST), it calls the EVENT.NUMBER and DISPLAY.INFO subprocedures and stops. Otherwise, it calls the ASK.FOR.DATE subprocedure.

LOOK.UP.DATE is the same as LOOK.UP.EVENT, but asks for a date rather than an event.

EVENT.NUMBER finds the number of the item on the event list that matches the event typed in. It then locates the date on the date list with that same number and represents the date by the variable name "DATE.

DATE.NUMBER finds the number of the item on the date list that matches the date typed in. It then locates the event on the event list with that same number and represents the event by the variable name "EVENT.

DISPLAY.INFO displays the event and date corresponding to the information requested. It then asks if the player wants to look up some other information. If so, it calls MEMO.PAD to repeat the activity. Otherwise, it ends the activity.

ASK.FOR.DATE displays a message saying it does not have the date asked for. It asks if it can add that date to its list. If the player types YES, it calls the ADD.ON.-DATE subprocedure. Otherwise, it calls MEMO.PAD to begin the activity again.

ASK.FOR.EVENT is the same as ASK.FOR.DATE, but asks if it can add an event rather than a date.

ADD.ON.DATE asks for the date to be added and calls the EXPAND.LISTS subprocedures to add that date to its date list. It then asks if the player would like to

look up something else. If the answer is YES, it calls MEMO.PAD to begin the activity again. Otherwise, it ends the activity.

ADD.ON.EVENT asks for the event to be added and calls the EXPAND.LISTS subprocedure to add that event to its event list. It then asks if the player would like to look up something else. If the answer is YES, it calls MEMO.PAD to begin the activity again. Otherwise, it ends the activity.

EXPAND.LISTS adds the new date and event to the computer's date and event lists.

```
TO MEMO.PAD
 TEXTSCREEN CLEARTEXT ASK
END

TO ASK
 PRINT [DO YOU WANT TO LOOK UP AN EVENT]
 PRINT [OR A DATE? (TYPE EVENT OR DATE.)]
 MAKE "TYPE REQUEST
 IF :TYPE=[EVENT] THEN LOOK.UP.EVENT STOP
 IF :TYPE=[DATE] THEN LOOK.UP.DATE STOP
 PRINT []
 PRINT [PLEASE TYPE ONLY EVENT OR DATE.]
 ASK
END

TO LOOK.UP.EVENT
 PRINT []
 PRINT [PLEASE TYPE IN THE EVENT YOU WANT]
 PRINT [ME TO LOOK UP.]
 MAKE "EVENT REQUEST
 IF MEMBER? :EVENT :EVENT.LIST THEN MAKE "NUM 0 EVENT.NUMBER :EVENT.LIST
   DISPLAY.INFO STOP
 ASK.FOR.DATE
END

TO LOOK.UP.DATE
 PRINT []
 PRINT [PLEASE TYPE IN THE DATE YOU WANT]
 PRINT [ME TO LOOK UP.]
 MAKE "DATE REQUEST
 IF MEMBER? :DATE :DATE.LIST THEN MAKE "NUM 0 DATE.NUMBER :DATE.LIST
   DISPLAY.INFO STOP
 ASK.FOR.EVENT
END
```

```
TO EVENT.NUMBER :EVENT.LIST
 MAKE "NUM :NUM+1
 IF :EVENT=FIRST :EVENT.LIST THEN MAKE "DATE ITEM :NUM :DATE.LIST STOP
 EVENT.NUMBER BUTFIRST :EVENT.LIST
END

TO DATE.NUMBER :DATE.LIST
 MAKE "NUM :NUM+1
 IF :DATE=FIRST :DATE.LIST THEN MAKE "EVENT ITEM :NUM :EVENT.LIST STOP
 DATE.NUMBER BUTFIRST :DATE.LIST
END

TO DISPLAY.INFO
 CLEARTEXT PRINT [ ]
 PRINT [THE EVENT] PRINT :EVENT
 PRINT [TOOK PLACE IN] PRINT :DATE
 PRINT [ ]
 PRINT [WOULD YOU LIKE TO LOOK UP]
 PRINT [SOME OTHER INFORMATION?]
 PRINT [(TYPE YES OR NO.)]
 MAKE "ANS FIRST REQUEST
 IF :ANS="YES THEN MEMO.PAD
 CLEARTEXT PRINT [YOU'VE EXITED MEMO.PAD.]
 TOPLEVEL
END

TO ASK.FOR.DATE
 PRINT [ ]
 PRINT [SORRY. I DON'T HAVE THAT]
 PRINT [INFORMATION IN MY FILE.]
 PRINT [CAN YOU GIVE ME THAT DATE]
 PRINT [SO I CAN ADD IT TO MY FILE?]
 PRINT [(TYPE YES OR NO.)]
 MAKE "INFO FIRST REQUEST
 IF :INFO="YES THEN ADD.ON.DATE STOP
 MEMO.PAD
END

TO ASK.FOR.EVENT
 PRINT [ ]
 PRINT [SORRY. I DON'T HAVE THAT]
 PRINT [INFORMATION IN MY FILE.]
 PRINT [CAN YOU GIVE ME THAT EVENT]
 PRINT [SO I CAN ADD IT TO MY FILE?]
 PRINT [(TYPE YES OR NO.)]
 MAKE "INFO FIRST REQUEST
```

```
 IF :INFO="YES THEN ADD.ON.EVENT STOP
 MEMO.PAD
END

TO ADD.ON.DATE
 PRINT [GOOD. WHAT IS THE DATE FOR]
 PRINT :EVENT
 MAKE "DATE REQUEST
 EXPAND.LISTS
 PRINT [THANKS. I'VE ADDED THAT ON]
 PRINT [TO MY LIST. WOULD YOU LIKE]
 PRINT [TO LOOK SOMETHING ELSE UP?]
 PRINT [(TYPE YES OR NO.)]
 MAKE "ANS FIRST REQUEST
 IF :ANS="YES THEN MEMO.PAD STOP
 CLEARTEXT TOPLEVEL
END

TO ADD.ON.EVENT
 PRINT [GOOD. WHAT EVENT TOOK PLACE IN]
 PRINT :DATE
 MAKE "EVENT REQUEST
 EXPAND.LISTS
 PRINT [THANKS. I'VE ADDED THAT ON]
 PRINT [TO MY LIST. WOULD YOU LIKE]
 PRINT [TO LOOK SOMETHING ELSE UP?]
 PRINT [(TYPE YES OR NO.)]
 MAKE "ANS FIRST REQUEST
 IF :ANS="YES THEN MEMO.PAD STOP
 CLEARTEXT TOPLEVEL
END

TO EXPAND.LISTS
 MAKE "DATE.LIST LPUT :DATE :DATE.LIST
 MAKE "EVENT.LIST LPUT :EVENT :EVENT.LIST
END
```

After defining these procedures by pressing <CTRL> C, type:

```
MAKE "DATE.LIST [[1215] [1492] [1776] [1803]]
```

and press <RETURN>. Then, type:

```
MAKE "EVENT.LIST [[MAGNA CARTA] [COLUMBUS DISCOVERS AMERICA]
 [DECLARATION OF INDEPENDENCE] [LOUISIANA PURCHASE]]
```

and press <RETURN>. Now, save the procedures and these two statements as a file on the Activity Disk by typing:

```
SAVE "MEMO.PAD
```

Chapter 17: WRONG.WAY

START.WRONG.WAY first prompts the player to put the Utilities Disk in the disk drive so that the MUSIC and SPRITES procedures can be read, and the VEHICLES shapes can be automatically loaded. WRONG.WAY then calls on the ERASE.SPRITES, SCREEN, ASSIGN.SHAPES, and POSITION.SHAPES subprocedures, sets the variables "HITS (the number of times your bicycle is hit by a car or a truck) and "MILES (the number of miles driven) to zero, and calls the DISPLAY and ANIMATE.SHAPES subprocedures.

ERASE.SPRITES sends all sprites back to the center of the screen, and hides them. It then makes the turtle the current sprite and shows it. SCREEN draws the screen background. ASSIGN.SHAPES sets the shapes and sizes of the sprites that are used in the activity. POSITION.SHAPES positions the sprites on the screen.

ANIMATE.SHAPES moves the sprites (in the shapes of cars and trucks) one at a time. For each sprite, it calls the MOVE subprocedure. Then it uses RC? to determine whether a key has been pressed, and, if one has, it calls the AVOID subprocedure. Finally, it checks to see if the distance traveled ("MILES) is equal to five. If it is, ANIMATE.SHAPES calls the RESULT subprocedure and ends the activity. Otherwise, it calls itself to begin again.

MOVE moves the current sprite forward a random number of steps between 15 and 25 and calls the WALL? and SPRITE? subprocedures.

BIKE? uses the TS? command to check whether a car or truck has hit the bicycle. If one has, BIKE? calls the SOUND.3 subprocedure to create the sound of a horn, increases the number of hits (:HITS) by one, moves the bicycle forward 20 steps, calls the DISPLAY subprocedure, and stops. Otherwise, it calls the SOUND.2 subprocedure to make a crash sound and stops.

AVOID moves the bicycle in the direction corresponding to the key pressed. If the bicycle is at the left edge of the screen, it increases the number of miles traveled ("MILES) by one and calls the DISPLAY subprocedure.

CHANGE moves the current sprite to a position on the screen slightly above center and gives it a random heading between 45 and 135 degrees.

SPRITE? uses TS? to determine whether two sprites are touching. If they are, it calls the BIKE? subprocedure, randomly sets the current sprite's heading to a value between 45 and 135 degrees, and moves the current sprite forward 50 steps.

WALL? checks whether the current sprite is close to the top or bottom border. If it is, WALL? calls the SOUND.1 and CHANGE subprocedures.

RESULT switches the screen to the TextScreen format, clears the screen, and displays a message saying how many times the bicycle was hit.

DISPLAY clears the text area of the screen, displays the number of miles traveled ("MILES), and asks if the player wants to move the bicycle.

SOUND.1, SOUND.2, and SOUND.3 use several commands from the MUSIC file on the Utility Disk to create sounds.

```
TO START.WRONG.WAY
 CLEARTEXT TEXTCOLOR 1 TEXTBG 6
 PRINT [PLEASE PUT THE LOGO UTILITIES DISK]
 PRINT [IN THE DISK DRIVE]
 PRINT []
 PRINT1 [PRESS <RETURN> TO CONTINUE]
 MAKE "ANS RQ
 PRINT []
 PRINT [READING MUSIC...]
 READ "MUSIC
 PRINT [READING SPRITES...]
 READ "SPRITES
 PRINT [READING VEHICLES...]
 READ "VEHICLES
 WRONG.WAY
END

TO WRONG.WAY
 ERASE.SPRITES DRAW BG 4 SCREEN
 ASSIGN.SHAPES POSITION.SHAPES
 MAKE "HITS 0 MAKE "MILES 0
 DISPLAY RANDOMIZE
 ANIMATE.SHAPES
END

TO ERASE.SPRITES
 MAKE "NUM 0
 REPEAT 7 [MAKE "NUM :NUM+1 TELL :NUM HT PU HOME]
 TELL 0 PU HOME ST
END

TO SCREEN
 HT PU PC 12 SETXY (-160) 90 SETH 90 PD
 REPEAT 20 [FD 320 LT 90 FD 1 RT 90]
 PU SETXY (-160) (-35) SETH 90 PD
 REPEAT 20 [FD 320 RT 90 FD 1 LT 90]
END
```

```
TO ASSIGN.SHAPES
 TELL 7 SETSHAPE 3 SMALLX SMALLY PC 1
 TELL 1 SETSHAPE 1 BIGX SMALLY PC 10
 TELL 2 SETSHAPE 1 BIGX SMALLY PC 7
 TELL 3 SETSHAPE 1 BIGX SMALLY PC 5
 TELL 4 SETSHAPE 2 BIGX SMALLY PC 11
 TELL 5 SETSHAPE 2 BIGX SMALLY PC 8
 TELL 6 SETSHAPE 2 BIGX SMALLY PC 14
END

TO POSITION.SHAPES
 MAKE "CUR 0
 MAKE "X (-150)
 MAKE "Y 80
 REPEAT 6 [MAKE "CUR :CUR+1 TELL :CUR SETXY (:X) (:Y) SETH ((RANDOM 91)+45) ST
  MAKE "X :X+35 MAKE "Y :Y-20]
 TELL 7 SETXY 125 30 SETH 270 ST
END

TO ANIMATE.SHAPES
 TELL 1 MOVE
 IF RC? THEN MAKE "KEY RC AVOID
 TELL 2 MOVE
 IF RC? THEN MAKE "KEY RC AVOID
 TELL 3 MOVE
 IF RC? THEN MAKE "KEY RC AVOID
 TELL 4 MOVE
 IF RC? THEN MAKE "KEY RC AVOID
 TELL 5 MOVE
 IF RC? THEN MAKE "KEY RC AVOID
 TELL 6 MOVE
 IF RC? THEN MAKE "KEY RC AVOID
 IF :MILES=5 THEN RESULT TOPLEVEL
 ANIMATE.SHAPES
END

TO MOVE
 FD (RANDOM 16)+10 WALL? SPRITE?
END

TO BIKE?
 MAKE "CUR SHAPE
 TELL 7
 IF TS? THEN SOUND.3 MAKE "HITS :HITS+1 FD 20 DISPLAY TELL :CUR STOP
 SOUND.2 TELL :CUR
END
```

```
TO AVOID
 TELL 7
 IF ALLOF :KEY="U YCOR < 80 THEN SETH 0 FD 10 SETH 270 STOP
 IF ALLOF :KEY="D YCOR > (−25) THEN SETH 180 FD 10 SETH 270 STOP
 IF :KEY="L THEN FD 10
 IF XCOR < (−150) THEN MAKE "MILES :MILES+1 DISPLAY
END

TO CHANGE
 SETY 30 SETH (RANDOM 91)+45
END

TO SPRITE?
 IF TS? THEN BIKE? SETH (RANDOM 91)+45 FD 50
END

TO WALL?
 IF ANYOF YCOR > 90 YCOR < (−25) THEN SOUND.1 CHANGE
END

TO RESULT
 CLEARTEXT ND PRINT [ ]
 PRINT [CONGRATULATIONS! YOU COMPLETED ALL 5]
 PRINT [MILES OF THE TRIP HOME AND ONLY GOT]
 (PRINT [HIT] :HITS [TIMES.] )
 PRINT [ ]
 PRINT [WITH ALL THAT PRACTICE I'LL BET YOU]
 PRINT [CAN DO EVEN BETTER NEXT TIME. JUST]
 PRINT [TYPE "WRONG.WAY" AND PRESS <RETURN>]
 PRINT [IF YOU WANT TO TRY AGAIN.]
END

TO DISPLAY
 CLEARTEXT
 (PRINT [MILES= ] :MILES [CRASHES= ] :HITS )
 PRINT [ ]
 PRINT [DO YOU WANT TO MOVE YOUR BICYCLE?]
 PRINT [LEFT=L; UP=U; DOWN=D]
END

TO SOUND.1
 WAVE 17 ATTACK 0 DECAY 15
 PLAY [9] [10]
END
```

```
TO SOUND.2
 WAVE 129 ATTACK 0 DECAY 15
 PLAY [0] [15]
 WAVE 17
END

TO SOUND.3
 WAVE 33 ATTACK 8 DECAY 15
 PLAY [5 -2 5 -2] [15 5 10 5]
 WAVE 17
END

MAKE "STARTUP [START.WRONG.WAY]
```

Index

A

Activity Time activities
  BIO, 296-99
  BUILDBLOCKS, 253-57
  FIND.ME, 69-73
  FRACTIONS, 185-86
  MAZE, 42-44
  MEMO.PAD, 345-47
  MY.WORD, 324-25
  RELAY, 230-31
  SPACETURTLE, 204-7
  SPELL'NDRAW, 103-5
  TEST.ME, 268-69
  TIC.TAC, 127-29
  WRONG.WAY, 366-67
Analytical geometry, 60, 65
Animation, 188, 196-202
  controlling movements, 196, 201-2
  demonstration files, 236
    DINOSAURS, 236
    RUNNER, 236
    SUBMARINE, 236
  drawing background scenes, 225-26
  procedures
    FLY.BALLOON, 220
    MOVE, 197-200, 202, 203
    SEA.SCENE, 222-27
  saving, 229
Arithmetic expressions, 261-63
  hierarchy of, 261-63, 264-65
  how to evaluate, 261-62
  use of parentheses in, 262
Arithmetic operations, 260-61, 263-67
  computations, 260-61
  displaying answer, 260
  keys used, 260
  in procedures, 263-67

B

BASIC, xiv, 7, 49, 93
Biorhythms activity (BIO), 296-99

C

Clearing computer memory, 100
Colors, 47-50
  background, 48-50
  pen, 48-50
  table of, 48
Combining text with graphics, 328-30
Commands
  ALLOF, 193-94
  ANYOF, 193-94
  BACKGROUND (BG), 48-50
  BACKWARD (BK), 26
  built-in (*see* primitive)
  BUTFIRST (BF), 316, 317, 318, 320
  BUTLAST (BL), 316, 317-18, 320
  CATALOG, 98
  CLEARSCREEN (CS), 38, 40
  CLEARTEXT (<SHIFT> and
      <CLR>), 40
  COS, 272, 274
  COUNT, 321
  <CTRL> A, 121
  <CTRL> B, 121
  <CTRL> C, 121
  <CTRL> D, 121
  <CTRL> F, 121
  <CTRL> G, 110-11, 120, 121
  <CTRL> K, 121
  <CTRL> L, 121
  <CTRL> N, 121
  <CTRL> O, 121
  <CTRL> P, 121
  <CTRL> W, 110-12
  <CTRL> Z, 110-12
  DOUBLECOLOR, 56-57
  DRAW, 29
  DRAWSTATE, 73-75
  EDIT (ED), 90, 117
  EMPTY?, 320
  END, 80-81
  ERASE (ER), 91, 92

Commands *(continued)*
ERASE NAMES (ER NAMES), 172
ERASEFILE, 102
ERASEPICT, 102
FIRST, 316-17, 318, 320
FORWARD (FD), 26-27
FPUT, 341-43
FULLSCREEN (<f5>), 39
GOODBYE, 100
HEADING, 75
HIDETURTLE (HT), 57
IF-THEN, 190-91, 193
IF-THEN-ELSE, 194-95
IFFALSE (IFF), 191-92
IFTRUE (IFT), 191-92
INTEGER, 272, 273
ITEM, 331, 332-35
LAST, 316-317, 320
LEFT (LT), 26
LIST?, 309, 310
LPUT, 341-43
MAKE, 166, 172
MAKE "STARTUP, 226-27
MEMBER?, 331-32
NUMBER, 309, 310
OUTPUT, 283-86
PENCOLOR (PC), 48-50
PENCOLOR −1 (PC −1), 50-51, 52
PENDOWN (PD), 38
PENERASE, 50-51, 52
PENUP (PU), 37, 40
primitive, 4, 26, 77-78, 100
PRINTOUT (PO), 90-91, 96
PRINTOUT ALL (PO ALL), 90-91, 97
PRINTOUT NAMES (PO NAMES), 171-72
PRINTOUT TITLES (POTS), 89
QUOTIENT, 272, 273
RANDOM, 272, 274
RANDOMIZE, 272, 274
READ, 101-2
READCHARACTER (RC), 196, 201-2
READPICT, 101
READSHAPES, 248
REMAINDER, 272, 273
REPEAT, 30-31, 32
REQUEST (RQ), 269, 313, 320
RIGHT (RT), 26, 27-28
ROUND, 272

Commands *(continued)*
SAVE, 95
SAVEPICT, 95, 97, 98, 229
SAVESHAPES, 248
SENTENCE (SE), 322-23
SETHEADING (SETH), 66-67
SETHEADING TOWARDS (SETH TOWARDS), 66-67
SETSHAPE, 218
SETX, 65, 69
SETXY, 60-69
SETY, 65, 69
SHAPE, 219
SHOWTURTLE (ST), 57
SIN, 272, 273
SINGLECOLOR, 56-57
SPLITSCREEN (<f3>), 39
SQRT, 272, 274
STOP, 202
TELL, 210
TEST, 191-93
TEXTBG, 52
TEXTCOLOR, 52
TEXTSCREEN (<f1>), 39
THING, 309
THING?, 309
TO, 79, 87
TOPLEVEL, 202
WHO, 219
WORD, 322-23
WORD?, 309-10
XCOR, 75
YCOR, 75
Commodore 64 computer system, *illustration*, 10-11
Computational commands
advanced, 272-75
COS, 274
INTEGER, 273
QUOTIENT, 273
RANDOM, 274-75
REMAINDER, 273
ROUND, 272
SIN, 273
SQRT, 274
Computational procedures
averaging (AVERAGE), 278-79
finding the greatest common divisor (GCD), 275-77
squaring (SQR), 278-80

Computational procedures *(continued)*
- transferring results, 283-88
- using recursion in, 281-82
- using variables in, 275-77

Computer-generated poetry, 339-41
Conditional commands, 191, 195, 196
- IF-THEN-ELSE, 194-95
- IFFALSE (IFF), 191-92
- IFTRUE (IFT), 191-92
- TEST, 191-92
- using more than one, 192-94

Constants, 131, 132
Coordinate geometry, 60, 65
Coordinate grid, 60-62
- commands
  - SETHEADING (SETH), 66-67
  - SETHEADING TOWARDS (SETH TOWARDS), 66-67
  - SETX, 65, 69
  - SETXY, 60-69, 73
  - SETY, 65, 69
- ordered pairs of values, 61
- positioning the turtle, 63-67
- X coordinate, 61-62
- Y coordinate, 61-62

Cursor, 13
- moving
  - in Edit mode, 81-84, 118-19, 121-22
  - in text, 16-19

D

Disk drive, 12
Disks
- blank, 93
- initializing, 78, 93
- inserting, 16
- loading from, 101-2
- Logo language, 11, 16, 78
- viewing catalogs of, 98-100

Drawing, 23-25
- in DoubleColor, 56-57
- in SingleColor, 56-57
- spirals using MAKE statement, 174-78
- squares, 28
- squares, using SETXY, 68-69
- switching between Draw and NoDraw modes, 25
- switching to Draw mode, 24
- with thicker lines, 56-57

Drawing commands. *See* Graphics commands

E

Edit mode
- correcting mistakes in, 81-84
- entering the, 78-79
- modifying procedures in, 117-22

Eraser, giving the pen an, 50-51
Error messages, 110

F

Files, 94
- animation, 236
- erasing, 102
- loading, 101-2
- naming, 94-95
- saving, 95
- self-starting, 226-27
- sound, 350
- sprite shapes, 211

Formats. *See* Screen formats
Formatting. *See* Initializing a disk
Fractions
- equivalent, using STAR procedure, 185-86
- reducing, 276

FullScreen, 38, 39, 50, 67

G

Gears. *See* "Objects to think with"
Geometry, analytical or coordinate, 60, 65
Grammar, English, parts of speech, 340
Graphs of trigonometric functions, 293-96
Graphics
- area of screen, 25
- clearing screen, 39-40
- combining with text, 328
- commands
  - BACKGROUND, 48-50
  - CLEARSCREEN, 38-40
  - CLEARTEXT, 40
  - DOUBLECOLOR, 56-57
  - DRAW, 29
  - DRAWSTATE, 73-75

Graphics *(continued)*
  commands
    FORWARD, 26-27
    FULLSCREEN, 39
    HEADING, 75
    HIDETURTLE, 57
    ITEM, 331, 332-35
    XCOR, 75
    YCOR, 75
  modes, 23-25
Greatest common divisor
  finding the, 275-77
  as a subprocedure, 276-77

I

IF-THEN statements, 191
Initializing a disk, 92-94
Input, 139
Integer, prime, 290-91
  printing a list of, 292-93
  testing if positive, 290-92

K

Keyboard, *illustration*, 14-15
  < ↑ >, 34-36
  <DEL> (Delete), 13, 18-19
  keys for correcting mistakes, 81-85, 86
  <RETURN>, 16
  <SHIFT>, 13, 16

L

Language disk, 11
List processing, 301-3
  combining words and lists, 322-24
  commands
    BUTFIRST (BF), 316, 317, 318, 320
    BUTLAST (BL), 316, 317-18, 320
    for displaying an item in a list, 332-35
    FIRST, 316-17, 318, 320-21
    for identifying an item in a list, 331-32
    ITEM, 332-33
    MEMBER?, 331-32
    for printing, 303-6
    for selecting parts of text, 316-22
    SENTENCE (SE), 322-23
Logo, 2-7, 11-13, 16-18
  advantages of, 4-7
  features of, 5-6
  loading, 16-18
  MIT Logo Group, 2
  origins of, 2-4
  starting, 11-13

M

MAKE statement
  drawing hexagons with, 169-70
  drawing octagons with, 171
  drawing spirals with, 174-79
  drawing squares with, 167-69
  and recursion, 281-82
Massachusetts Institute of Technology, 2-3, 5
  MIT Logo Group, 2
  Seymour Papert, 2, 3, 4, 5
Math phobia, 4
"Mathland environment", 3-4
Memory. *See also* workspace
  clearing, 100
Modes
  Draw, 23-24, 25
  Edit, 78-81
  NoDraw, 23-24, 25
  NoWrap, 32
  Wrap, 32
Modular programming, 4, 5, 6
Monitor, 12
Music. *See* Sound

N

NoDraw mode, 23-24, 25
NoWrap mode, 32

O

"Object to think with", 2-3, 25, 209

P

Papert, Seymour, 2-4, 5
Parallelogram, 141
Pascal, 5
Pixels, 238-39, 241-44, 246-48
Playing turtle, 41

Poetry, computer-generated, 339-41
Polygon patterns
  drawing hexagons, 146-47, 169-70
  drawing stars, 154-58
  drawing tiles, 150-52
  drawing wreaths, 147-49, 164-65
  using POLY procedure, 142-45
  using variables, 142-52
Primitive commands, 4, 26
  use in procedures, 77, 78
Procedures
  ADD, 276
Procedures
  ATTACK, 364
  AVERAGE, 278-80
  BACK.SLOWLY, 267, 268
  BIGGER.SQUARES, 195-96
  calling, 188
  CIRCLE, 88-89
  COMPOSE, 358-62
  COUNT.ALT, 264-67
  COUNT.CHAR, 319-20
  COUNT.DOWN, 267
  COUNT.SLOWLY, 267, 268
  COUNT.UP, 263-67
  DECAY, 364
  defining, 79-81, 85, 87
  DISPLAY.SPRITES, 214-15, 216
  DOUGHNUT.CIRCLES, 116-17
  DOUGHNUT.SQUARES, 115
  DOWN.ALT, 267
  DRUM, 354-56, 361
  ending, 80-81
  ERASE.SPRITES, 215
  erasing, 91
  error message, 110
  EXP, 281-82
  FEED.TIME, 328-30
  FILL.PARA, 141
  FILL.RECT, 141
  FILL.RHOMBUS, 138
  FILL.SQUARE, 136-37
  FLY.BALLOON, 220
  FRIEND?, 333-35
  GRAPH, 293-96
  GUESS.MY.NUMBER, 289-90
  HIDE.AND.SHOW, 126
  interactive, 310, 313-15
  KEEP.SQUARING, 188-89
  LETTER, 312-13

Procedures *(continued)*
  levels of, 110
  LIST.FAC, 286-88
  listing, 89
  modifying in Edit mode, 117-22
  MULTIPLY, 278, 281-83
  naming, 77, 87-88
  OCT.TILE, 150-52
  PARA, 141
  PENT, 147-49
  personalizing with REQUEST
    command, 313-15
  PLAY, 356-58
  POLY, 142-46, 154-56
  PRIME?, 291-92
  PRINT.PRIMES, 292-93
  printing, 90-91
  PULSE, 365
  PYTHAG, 284-85
  RECT, 139-41
  RECTANGLE, 88
  RELEASE, 365
  REVERSE, 323-24
  running (*see* using)
  saving, 91-92
  SCALE, 358
  SHAPES, 114
  SLOWDOWN, 351
  SONG, 359-60
  SQ.OUT, 189-91
  SQ.RAINBOW, 178-79
  SQ.SPIRAL, 174-78
  SQR, 278-81
  SQUARE, 78-79, 80-81, 83-86
  SSH, 350-51
  SSHER, 351-53
  STAR, 155-59
  STAR modified, 159-63
  STAR.RAINBOW, 180-81
  STAR.SPIRAL, 179-80
  stopping, 110-12
  subprocedures, 108-10
  SUBTRACT, 277-78
  SUSTAIN, 365
  TEMPO, 353-56
  that "learn", 341-43
  that "remember", 343-45
  TRANSLATE, 335-37
  TRI, 135-36
  TRI.SPIRAL, 181-84

Procedures *(continued)*
TRIANGLE, 88
using, 86-87
using commands within, 122-25
using constants in, 131, 132
using variables in, 131, 133-37
viewing, 89
WAVE, 363-64
within procedures, 108-10
WREATHS, 164-65
writing, 78-79

R

Recursion, 187-89
animating sprites with, 220-25, 227-29
computational procedures with, 281-82
conditional statements with, 195-96
drawing square-spiral design with, 189
KEEP.SQUARING, 188-89
printing text with, 311
stopping, 190-91
using the MAKE statement with, 281-82
Rhombus, 137-38

S

Screen. *See also* Monitor
as grid, 59-62
Screen formats
FullScreen, 38, 39, 50
SplitScreen, 38-39, 50
TextScreen, 38, 39, 50
Screen pictures
loading, 101-2
saving, 95, 97, 98
Shapes. *See* Sprites
Sound
attack and decay, 364
characteristics of, 350
COMPOSE, 358-62
effects, 366-67
modifying the quality of, 362-65
MUSIC file on Utilities Disk, 349-65
Sound *(continued)*
procedures on Utilities Disk, 350-65
ATTACK, 364
DECAY, 364
PLAY, 356
PULSE, 365
RELEASE, 365
SSH, 351
SSHER, 351-52
SUSTAIN, 365
TEMPO, 353-54
WAVE, 362-63
pulse, 365
sustain and release, 365
using Logo Utilities disk, 349-50
varying the duration of, 350-53
varying the pitch of, 356-58
varying the tempo of, 353-56
waveforms, 362-64
writing a song, 358-62
Spirals, drawing
successive parts of, 178-79
using geometric figures, 179-81
using MAKE statement, 174-77
SplitScreen, 38-39, 50
Sprites, 209-10
BUILDBLOCKS activity, 253-57
commands
SETSHAPE, 218, 219
SHAPE, 219
TELL, 210
WHO, 219
creating, 241-43, 246
detecting collisions of, 248-53
editing predefined, 236-41
files on Utilities Disk, 211-13
ANIMALS, 211
ASSORTED, 213
RUNNER, 213
SHAPES, 212
VEHICLES, 212
finding current number and shape files on Utilities Disk, 219
giving them shapes, 211-14
loading, 214
moving to center of screen, 215
procedures
BIGX, 234
BIGY, 234
SMALLX, 234-35

Sprites *(continued)*
RELAY activity, 230-31
saving, 244
shapes represented by number and name, 216-18
stretching, 234-36
Sprite Editor
box, 238-39, 246-48
cursor, 238, 243
mode (EDSH), 236-37, 239-42
Squares, 28
with alternating colors, 53
filling, 55
filling with color, 53-54
repetition, 112-13
using SETXY, 68-69
using a variable, 133-37
Star, drawing a, 154-61
using POLY procedure, 154-161
using STAR procedure, 155-65
Subprocedures, 108-10

T

Temperatures, procedures to convert, 280
Text. *See also* List processing
to change color, 52
clearing, 39-40
combining with graphics, 328
combining lists, 322-23
combining words, 322-23
letter, sample, 312-13
lists defined, 302-3
printing, 303-6
using recursion, 311
words defined, 302-3
Text area of screen, 25, 38-39
Translating words, 335-37
Triangles, 113, 135
repetition of, 113
using a variable, 135
Turtle, 25
as sprite, 209-10
color of, 47, 48
hiding, 57
Home position of, 59, 61, 66
origin of, 3
positioning on X,Y grid, 63-67
rotating, 66-67

Turtle *(continued)*
showing on screen, 57
status of, 73-75
width of trail, 56-57

U

Utilities Disk
demonstration files on, 236
sound procedures on, 349-50
SPRED file on, 236-37
sprite procedures on, 211, 214, 216, 220, 233-34
sprite shapes files on, 211

V

Variables, 131, 133-37
assigning words and lists as, 307-8
checking current status of, 308-11
commands
LIST?, 309-10
NUMBER?, 309-10
THING, 309
THING?, 309
WORD?, 309-10
defining, 166
determining value for, 308-11
erasing, 172
names, 133
using MAKE statement, 166, 173
using more than one, 139-41
viewing, 171-72
Vocabulary extension, 4

W

Work Disk, 93-94, 127
amount of disk space, 99-100
asking for catalog, 98-100
saving background on, 225-26
saving sprites on, 244-45
Workspace, 94, 98, 100, 102
Wrap mode, 32

X

X,Y coordinate grid system, 61-65, 73
ordered pairs of values, 61

Kenneth P. Goldberg

At a time when Logo is becoming increasingly important in the classroom, Kenneth P. Goldberg has established himself as a national authority on the subject. He earned both his B.A. and M.S. degrees in Mathematics at New York University and his Ph.D. in Mathematics at Michigan State University. In 1973, he joined the faculty at New York University and has since become Professor of Mathematics Education and Chairman of the Department of Mathematics, Science, and Statistics Education. Kenneth's credentials as a writer and educator are impressive: he developed and now teaches a course for educators on Logo; he has written articles for *Family Computing* magazine and for an upcoming edition of the *Encyclopaedia Britannica;* and he was, for a time, the editor of *The New York State Mathematics Teachers' Journal.* He is also the author of *The Parents' Book on Calculators* and *Pushbutton Mathematics,* and the co-author of *Microcomputers A Parents' Guide.* Born in Brooklyn, Kenneth now lives in upstate New York.

The manuscript for this book was prepared on an Apple II personal computer. Submitted to Microsoft Press in electronic form, the text files were processed and formatted using Microsoft Word.

Cover and text design by Ted Mader and Associates. Cover and interior illustrations by Mits Katayama. Computer drawings by Rick van Genderen. Raster graphic image displays produced on the Commodore 64 computer, photographed by Ed Lowe.

Text composition in Palatino, with display in Helvetica Black, using CCI Book, and the Mergenthaler Linotron 202 digital phototypesetter.

Cover art separated by Color Masters, Phoenix, Arizona; printed on 12 pt. Carolina by Philips Offset Company, Inc., Mamaroneck, New York. Text stock, 60 lb. Glatfelter Offset, supplied by Carpenter/Offutt. Book printed and bound by Fairfield Graphics, Fairfield, Pennsylvania.

Other Titles from Microsoft Press

THE IBM ENVIRONMENT

Running MS-DOS
The Microsoft Guide to Getting the Most Out of the Standard Operating System for the IBM PC and 50 Other Personal Computers
Van Wolverton
ISBN 0-914845-07-1 *$19.95*

Exploring the IBM PCjr
Peter Norton
ISBN 0-914845-02-0 *$18.25*

Discovering the IBM PCjr
Peter Norton
ISBN 0-914845-01-2 *$15.95*

Managing Your Business with Multiplan
How to Use Microsoft's Award-Winning Electronic Spreadsheet On Your IBM PC.
Ruth K. Witkin
ISBN 0-914845-06-3 *$17.95*

Getting Started with Microsoft WORD
A Step-by-Step Guide to Word-Processing
Janet Rampa
ISBN 0-914845-13-6 *$16.95*

THE APPLE ENVIRONMENT

The Apple Macintosh Book
Cary Lu
ISBN 0-914845-00-4 *$18.95*

Presentation Graphics On the Apple Macintosh
How to Use Microsoft Chart to Create Dazzling Graphics for Professional and Corporate Applications
Steve Lambert
ISBN 0-914845-11-X *$18.95*

MacWork MacPlay
An Assortment of Creative Ideas for Fun and Profit
Lon Poole
ISBN 0-914845-22-5 *$18.95*

The Endless Apple
How to Maintain State-of-the-Art Performance On Your Apple II and IIe
Charles Rubin
ISBN 0-914845-27-6 *$15.95*

Learning Apple Logo Together
An Activity Book for Creative Parents, Teachers, and Kids
Kenneth P. Goldberg
ISBN 0-914845-25-X *$14.95*

GENERAL

Silicon Valley Guide
David Remer, Paul Remer, and Robert Dunaway
ISBN 0-914845-09-8 *$19.25*

Available wherever fine books are sold.

Kenneth P. Goldberg

For all parents and teachers who want to work with their children to develop skills that will enrich their education and expand their imaginations, here is an authoritative, enjoyable guide to understanding and exploring Logo—widely considered the best programming language for children. Dr. Kenneth Goldberg, a noted authority on Logo, provides an enlightening introduction to what Logo is and why it was created. Included are lively activities for children of all ages, that take advantage of all the colorful graphics and exciting sound capabilities of Commodore 64 Logo. All activities are carefully designed to complement the child's classroom activities—in arithmetic, geometry, trigonometry and English—and feature special Turtle Tips sections to help the parent or teacher guide the child through the more complicated features of Logo.

Dr. Goldberg is currently the Chairman of the Department of Mathematics, Science and Statistics Education at New York University. In addition to teaching Logo courses for educators, he writes for *Family Computing* Magazine.

MICROSOFT PRESS

$14.95 1084 S&S No. 0-671-30930-7 ISBN 0-914845-24-1