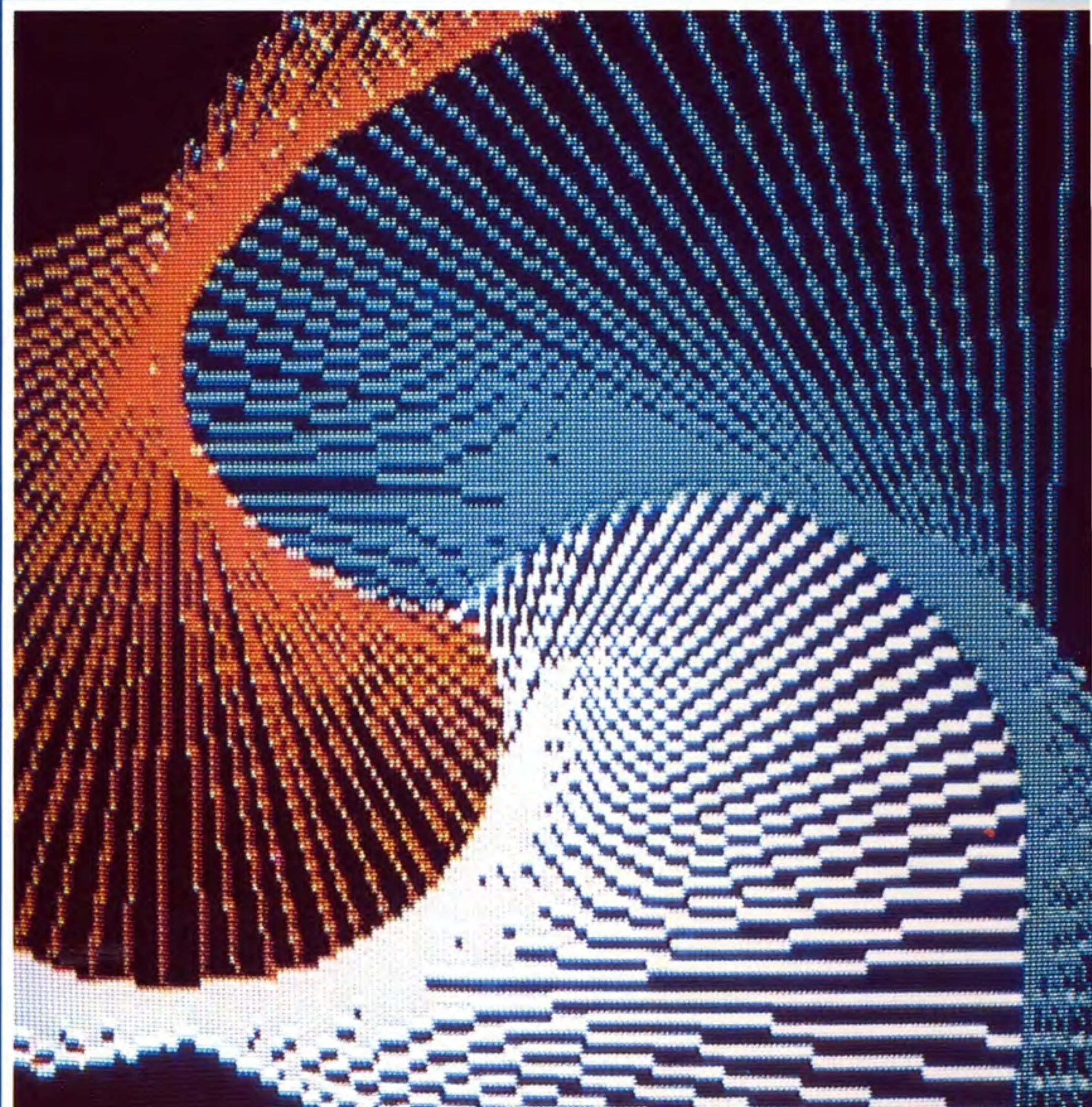
D.L. LOGO



TERMS AND CONDITIONS OF SALE AND LICENSE OF TANDY COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND RADIO SHACK FRANCHISEES OR DEALERS AT THEIR AUTHORIZED LOCATIONS

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this computer hardware purchased (the "Equipment"), and any copies of software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment. RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. **This warranty is only applicable to purchases of Tandy Equipment by the original customer from Radio Shack company-owned computer centers, retail stores, and Radio Shack franchisees and dealers at their authorized locations**. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or a participating Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.

D. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK MAKES NO EXPRESS WARRANTIES, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS LIMITED IN ITS DURATION TO THE DURATION OF THE WRITTEN LIMITED WARRANTIES SET FORTH HEREIN.**

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

A. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE." IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE."**
NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the TANDY Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.

G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby Radio Shack sells or conveys such Equipment to a third party for lease to CUSTOMER.

B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and or licensor of the Software and any manufacturer of the Equipment sold by Radio Shack.

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

12/84

D.L. LOGO

10 9 8 7 6 5 4 3 2 1

CONTENTS

INTRODUCTION TO DL LOGO 1

System Requirements (equipment you need) 3
Backing Up Diskettes 4
Starting the D.L. LOGO Program 5
LOGO in Action (a demonstration program) 6
The LOGO Philosophy 8
About this Manual 10
A Look at LOGO (color plates of LOGO screen displays) 11

1/ SIGHTSEEING (introductory graphics) 21

The Keyboard (keyboard functions with LOGO) 22
LOGO and the Primitives (LOGO commands) 25
Proceeding (multiple commands) 28
Abbreviations and Directions 30
Play it Again (repeating commands) 32
Bye (exiting LOGO) 34

2/ TECHNICOLOR TURTLE (using colors) 39

Turtle Shows Its True Colors (selecting pencolors) 40
Background Colors 42
Calculating Colors 43
Housekeeping (clearing the graphics and text screens) 45

3/ RUNNING ERRANDS (procedures for your Turtle) 53

Writing Procedures (a look at the edit mode) 54
About Editing (changing procedures) 57
Direct Wire (information from the keyboard) 64

CONTENTS

4/ MEMORIES, MEMORIES 73

Saving and Loading (using diskette storage) 74
Making Room (deleting files) 78
A Vanishing Act (erasing and restoring files) 80
Printing Procedures 84

5/ COMPLETING THE TOUR (more Turtle graphics) 91

More Geography (advanced graphic commands) 92
More Turtle Maneuvers 95
Building Fences and Roaming Free (Turtle screen control) 100

6/ YOU SING, TURTLE PLAYS (Turtle music) 107

Technical Talk 108
Getting Started 111
Everybody Sing (music in 4 voices) 122

7/ VARIETY: THE SPICE OF LIFE (using variables) 133

Making Things: Creating Words and Lists 134
Handling Data (what a variable is) 138
Counting on Variables (number variables) 144
Variables at Home and Abroad (local and global variables) 148
Ins and Outs of LOGO (output) 150

8/ TURTLE'S JUGGLING ACT (more about variables) 157

Words, Numbers, and Lists 158
Making a List 160
Talking in Sentences (using SENTENCE) 162
The First Shall Be Last (manipulating variables) 163
Running Away with Lists 175
Turtle Swallowed a Clock (using the time function) 177

9/ A CALCULATING TURTLE (working with numbers) 185

The Numbers Game 186
Turtle Figures 188
Comparing Figures and Facts 201
Calculating with Precision 205
Making Rearrangements (random, shuffling, sorting) 206

CONTENTS

10/ LOGIC AND LOOPS (loops and conditional logic) 221

Testing, Testing, 1, 2, 3 222
A Loop Is a Loop Is a Loop Is a 230
A Recursive Turtle 236

11/ TALKING BACK 245

You Write the Speech (using speech) 246
A Sound Procedure (LOGO's SOUND primitive) 250

12/ TURTLE ON A LEASH (joystick and X-Pad control) 259

Reading the Joysticks 260
Padding about, Turtle Style (using the X-Pad) 266
Remote Control 270

13/ CHASING AFTER BUGS (finding and handling errors) 275

To Err is Human (what errors are) 276
Debug (finding errors) 277
The Catch (using errors) 280
Trace—How To Follow a Procedure 285
Error Code Reference 287

14/ TURTLE IN THE DRIVER'S SEAT (files, OS-9, screen format) 295

Breaking Out (file handling procedures) 296
Fine Tuning File Control 301
Calling on OS-9 306
Turtle on The Textscreen (text screen formatting) 307

15/ EDITING: THE FULL STORY 319

Editing Features (advanced editing) 310
The Search (search and replace) 326
Edit Commands Reference 328

APPENDIX A/ Glossary of Terms for D.L. LOGO 335

APPENDIX B/ Sample D.L. LOGO Programs 347

APPENDIX C/ Primitive Reference 373

APPENDIX D/ Starting OS-9 from BASIC 391

Index 393

INTRODUCTION TO D.L. LOGO

Information about LOGO

Section 1	System Requirements: information about the equipment you need to use D.L. LOGO.
Section 2	Setting Up Your System: information about organizing your equipment to use D.L. LOGO.
Section 3	Backing Up Diskettes: information about backing up D.L. LOGO diskettes.
Section 4	Starting the D.L. LOGO Program: information about *booting* (starting) D.L. LOGO.
Section 5	LOGO in Action: a demonstration program to introduce D.L. LOGO.
Section 6	The LOGO Philosophy: the purpose and use of LOGO.
Section 7	About This Manual: the philosophy and organization of your D.L. LOGO manual.
Section 8	A Look at LOGO: color plates of LOGO-created display screens.

Introduction

LOGO is an exciting computer language designed to help students of any age explore structural and logical concepts. It employs what is often known as *turtle graphics*. The character that flashes around your video screen, drawing designs at your command, is called a *turtle*.

The reason for the name goes back to the early days of computer development when technicians directed an actual mechanical device moving at slow speed around laboratory floors.

Although the use of computers in education is still in its infant stage, major advancements have already been made in applying this revolutionary medium for learning activities. The LOGO language is one noteworthy application.

D.L. LOGO, running under the OS-9 operating system, is an advanced version of this special *graphics* language. It incorporates such features as:

- A graphics mode that lets you enter commands and see immediate execution
- A total of 16 background and 16 foreground screen colors
- The capability of interfacing with external devices, such as a robot, an external turtle, or a plotter
- Sophisticated speech, music, and sound capabilities
- Joystick, X-Pad, and printer interfacing
- The ability to accept variable input and to transfer variables between procedures
- The ability to access and use OS-9 commands and capabilities
- Advanced arithmetic, trigonometric, and logic functions
- Extremely high precision math capabilities
- Advanced sentence, word, and list manipulation
- Sound and speech capabilities

. . . . About Turtle Talk

Throughout this manual are special notes in the margins. These notes provide capsules of LOGO concepts or extra information to guide you through D.L. LOGO.

*. . . . **About Colors***

Because D.L. Logo's graphics' capabilities make extensive use of colors, a black and white television displays some designs poorly and may be unsatisfactory for some applications.

Section 1 System Requirements

MINIMUM EQUIPMENT

To use D.L. LOGO, you need:

- A 64K Color Computer
- One disk drive
- A television set
- The D.L. LOGO program diskette

OPTIMAL EQUIPMENT

To use all D.L. LOGO functions, you need:

- A 64K Color Computer
- Two disk drives
- A color television
- The D.L. LOGO program diskette
- A Multi-Pak Interface
- A Color Computer Speech/Sound Cartridge
- An X-Pad Graphics Tablet
- Two Color Computer Joysticks
- A printer. Excellent choices are a dot matrix printer, such as the Tandy DMP 410, or an ink jet printer, such as the Tandy CGP-220

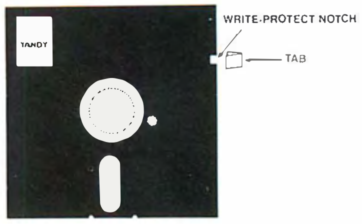
To give speech capabilities to D.L. LOGO, you need both the Multi-Pak Interface and the Speech/Sound Cartridge. The Speech/Sound Cartridge generates speech. The Multi-Pak Interface lets you connect both the disk drive(s) and the Speech/Sound Cartridge to your computer. To use the X-Pad Graphics Tablet, you also need the Multi-Pak Interface. Insert the Color Computer Disk Controller into Slot 4 of the Multi-Pak. The Sound/Speech Cartridge and the X-Pad Cartridge can then be inserted in any of the remaining slots. Set the Multi-Pak slot selection switch to 4 to access your disk drive cartridge. You do not need to change the selection switch to use other cartridges with D.L. LOGO.

Section 2 Backing Up Diskettes

You need to make 1 or more backup copies of D.L. LOGO before using your program diskette. You can make copies of any D.L. LOGO diskette in either of 2 ways.

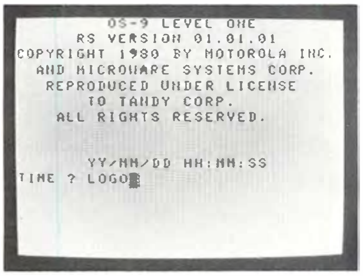
If you do not have the OS-9 operating system, you can back up diskettes using the normal Color Computer Disk BASIC, DSKINI, and BACKUP commands. Follow exactly the same backup procedures as outlined in your *Disk System Owner's Manual.*

Because D.L. LOGO operates under the OS-9 operating system, all of OS-9's features and commands are available. Using OS-9 commands, you can back up diskettes, make copies of files, or change directories. You can do these and other OS-9 operations either before you enter D.L. LOGO or from the LOGO program. Refer to your *OS-9 Commands* manual and Chapter 14 of this manual for information on using system commands.

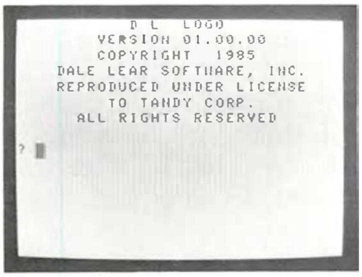


. . . . *About Backup*

Often the content of a diskette represents many hours of work. Losing this data can be discouraging. We suggest you make frequent copies of all your diskettes as you investigate and use D.L. LOGO. The process only takes a few minutes. Before making copies of your D.L. LOGO diskette, place a tab over the write-protect notch. When the backups are finished, store the original diskette in a safe place.



```
       OS-9 LEVEL ONE
     RS VERSION 01.01.01
COPYRIGHT 1980 BY MOTOROLA INC.
   AND MICROWARE SYSTEMS CORP.
    REPRODUCED UNDER LICENSE
         TO TANDY CORP.
     ALL RIGHTS RESERVED.

       YY/MM/DD HH:MM:SS
TIME ? LOGO
```



```
           D L   LOGO
        VERSION 01.00.00
        COPYRIGHT   1985
    DALE LEAR SOFTWARE, INC.
    REPRODUCED UNDER LICENSE
         TO TANDY CORP.
      ALL RIGHTS RESERVED
?
```

Section 3
Starting the D.L. LOGO Program

Use the OS-9 boot procedure that you are accustomed to using. If you do not have a standard OS-9 boot procedure, see Appendix D in this manual.

After starting up your system, the OS-9 BOOT and copyright messages are displayed on the screen. The following prompts appear:

```
   YY/MM/DD      HH:MM:SS
TIME ?
```

1. Enter the date in the year/month/day format, press the space bar, then enter the time as shown, and press ENTER.

   Note: Entering the time is optional. To bypass it, press ENTER after entering the date.

2. When the `OS-9:` prompt appears on the screen, type:

   `LOGO` ENTER

3. The D.L. Logo copyright appears on the screen. You are in the immediate, or single command mode and can begin entering commands.

Section 4
LOGO in Action

One of the best things about owning a new product is to see it working well. To let D.L. LOGO show off, leave your backup diskette in Drive 0, and type the following as your first command:

```
    LOAD "DEMO [ENTER]
```

The blue cursor disappears, and Drive 0 runs for a few moments. When the blue cursor reappears on the screen (your screen shows a question mark (?) followed by a blue block), type:

```
    DEMO [ENTER]
```

If you have a Speech/Sound cartridge installed, the demonstration program greets you audibly, as well as on the screen display, and speaks to you throughout the demonstration. Sit back and enjoy the display. When the program ends, it repeats itself. To stop the program, press [BREAK].

All the programs used in the demonstration are on your D.L. LOGO diskette. Following is a complete list or directory of all the programs provided with D.L. LOGO.

D.L. LOGO Directory

```
ANIMAL*       ANIMALS       SONGS*
HEX*          TREE*         FLAG*
CLOCK*        SORT*         FACTORIAL*
DEMO          FORTRESS      AITEXT
STARS*        SOUNDTEXT     GRAPHTEXT
INTROTEXT     MATHTEXT      EXITTEXT
DEMOSONG      DOODLE*
```

Programs not marked with an asterisk are part of the

. . . . About Keys

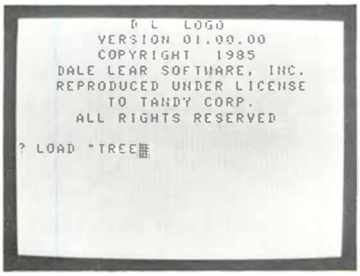
When you see a word enclosed in a box, such as [ENTER], press the key on your keyboard that matches the enclosed word. When you see [ENTER], press the ENTER key on your keyboard. When you see [SHIFT], press the SHIFT key.

. . . . About Filenames

Some of the filenames on your D.L. LOGO diskette end with the letters "TEXT". These files are text files rather than procedure files. You can load them into the D.L. LOGO workspace to examine or edit them, but you cannot execute or run them.

. . . . *About Procedures and Programs*

Procedures are a series of commands that direct LOGO to accomplish a task. Often, you use several procedures together to achieve a particular goal. Procedures used together make up a program.



```
          D L   LOGO
      VERSION 01.00.00
      COPYRIGHT   1985
   DALE LEAR SOFTWARE, INC.
  REPRODUCED UNDER LICENSE
       TO TANDY CORP.
     ALL RIGHTS RESERVED

? LOAD "TREE
```

demonstration program and do not function properly alone. Programs marked with an asterisk can be loaded and executed individually. To load and execute a program, follow the same procedure as you did for the demonstration program. Type:

LOAD "*filename* [ENTER]

Where *filename* is the name of the program you wish to load. For instance, to load the program TREE, type **LOAD "TREE** [ENTER].

To execute the program, type its name and press [ENTER]:

TREE [ENTER]

Some of the programs on your D.L. LOGO diskette require more than 1 word to execute. Appendix B contains listings and instructions for all the programs. There are also programs in Appendix B that are not included on the D.L. LOGO diskette. To use these programs, type them into D.L. LOGO's memory. As you learn more about LOGO, you may wish to re-examine these listings.

Now it is your turn to put D.L. LOGO through its paces. Try the concepts in this manual, and learn as you go.

When you finish, you can create programs even more exciting and interesting. Relax, have fun, and enjoy the language of D.L. LOGO.

Section 5 The LOGO Philosophy

LOGO is a language for learning. Children and adults have a natural fascination for LOGO's screen graphics capabilities. Add the functions of math, logic, speech, music, and sound, and LOGO becomes an exciting method for exploring new concepts at multiple levels.

LOGO is for play. You can weave shapes, colors, relationships, music, and sound in infinite patterns. Users unfamiliar with LOGO's language and concepts can begin with the simplest of structures. Later, they can increase intricacies and variety.

LOGO is a challenge for both adults and children. LOGO handles Boolean logic and advanced trigonometry as readily as simple arithmetic.

LOGO is for experimentation. Goals are important, but more important are the paths you explore in questing a goal. A square becomes an elaborate geometric design. A mistake in logic introduces a new adventure or, possibly, a new logic.

A Word to Adults

Although LOGO is an excellent educational tool for children, it can also stretch an adult's logic and conceptual abilities to the limit. Use LOGO to transform geometric and trigonometric concepts into immediate visual displays or to doodle, explore, and enjoy the fascination of unexpected results.

Wielding the techniques you learn in this manual, you can introduce toddlers and teenagers alike to the wonder of computer graphics and the excitement of having a machine respond to them.

. . . . About Logo

The philosophy of Logo as an educational tool was first advocated by Seymour Papert at the Massachusetts Institute of Technology. Papert suggested that Logo could provide a means whereby an individual could gain control over his educational progress. Rather than being taught by the computer, the student could teach the computer. Logo makes the computer a tool, "an object to think with," Papert said, rather than a master.

. . . . About Logo and Adults

Due to its graphics capabilities and ease of use, Logo was first considered to be a computer language for children. However, Logo has since developed and expanded. Newer versions of Logo have capabilities previously associated only with more sophisticated languages. At the same time, Logo graphic capabilities have improved. All this has been accomplished without loss of the basic simplicity of the Logo language. Because of these features, D.L. Logo is an ideal way for both children and adults to begin computer programming. The concepts you learn in this book are applicable in all other computer languages.

D.L. LOGO is far from being just another graphics program. This version of LOGO is a sophisticated computer language, capable of creating and manipulating data structures and files, performing advanced calculations, and interfacing with several peripheral devices.

A Word to Students

LOGO enables you to do some very special things on your computer. Don't let the length or size of your D.L. LOGO manual discourage you. LOGO is as easy or as hard to use as you want it to be.

If you want LOGO to be simple and fun to use, it is. If you want LOGO to challenge and test your mind, it does.

This book is full of examples and ways to use LOGO. Feel free to skip through it and find the things that interest you. Explore, experiment, have fun, and, above all, **don't** expect LOGO to be work.

Section 6 About This Manual

This manual guides you through D.L. LOGO. It begins with those concepts that are easiest to understand. More challenging concepts and projects come in later pages.

Similarly, to make the manual useful to all ages, most chapters progress from easy to more difficult programming examples. If you find a chapter is getting too complicated, move on to the next. As you learn more about LOGO, you can come back to the difficult sections.

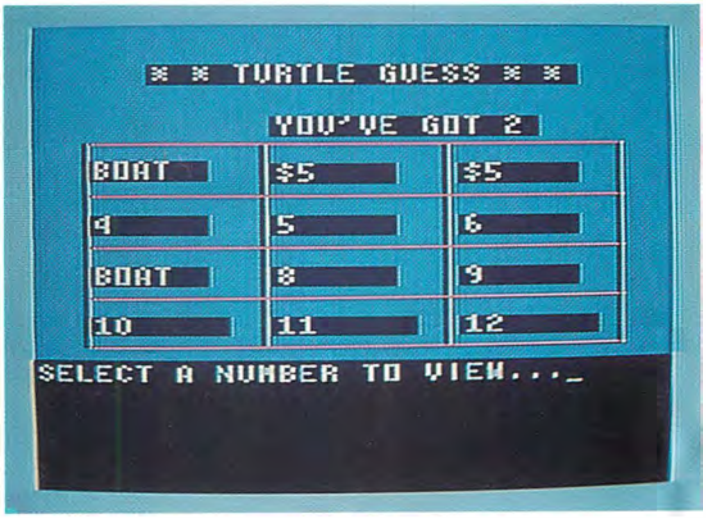
However, because many LOGO concepts interact, a strict progress from easy to hard is not always possible. Pursue those areas that most interest you. Learning one concept often helps you learn others.

Examples of procedures and programs illustrate LOGO concepts. Most of these examples are short and simple. They are easy to type and understand. Occasionally, the manual includes longer procedures and programs to demonstrate more fully the power of LOGO.

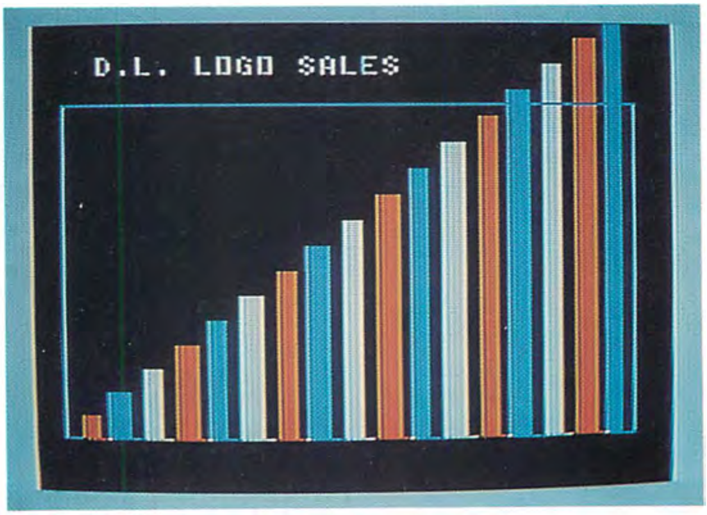
End-of-chapter summaries and suggested projects help you remember the ideas presented in each chapter. Using LOGO immediately is the quickest way to learn.

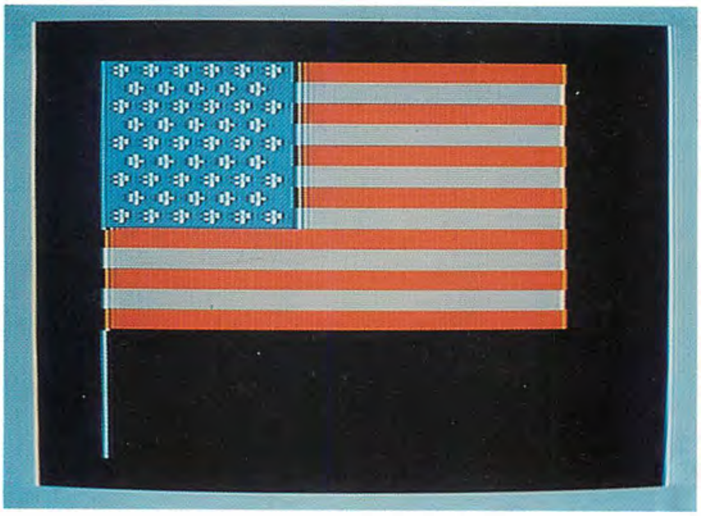
Section 7
A Look At Logo

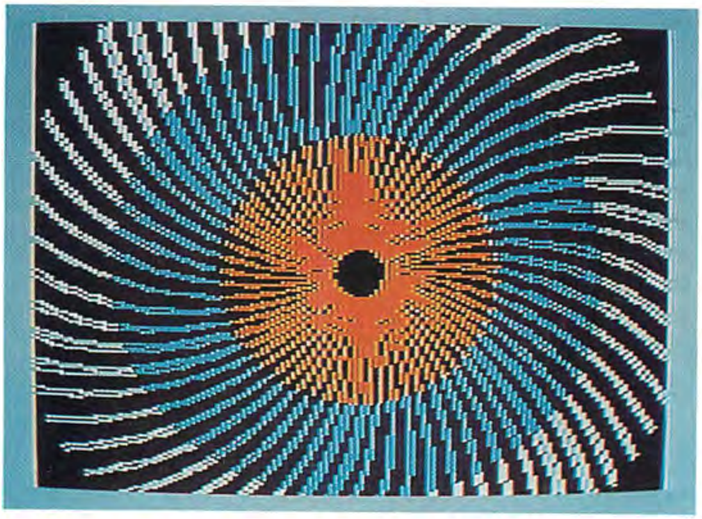
The following photographs show D.L. LOGO in use.

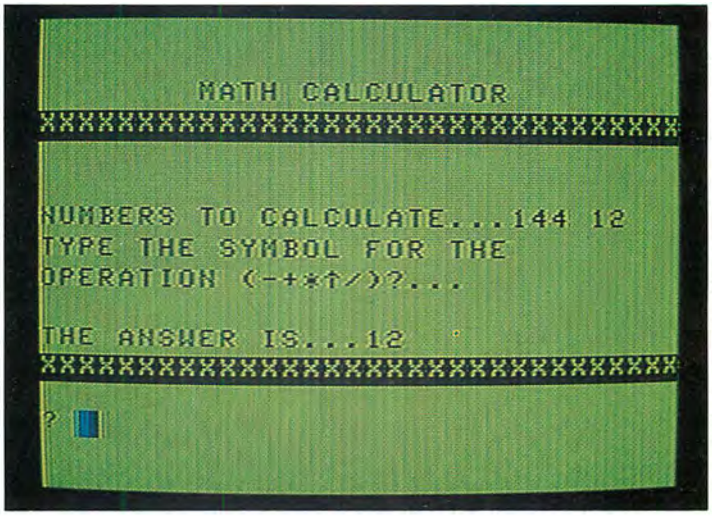


```
      * * TURTLE GUESS * *
            YOU'VE GOT 2
BOAT          $5            $5
4             5             6
BOAT          8             9
10            11            12
SELECT A NUMBER TO VIEW..._
```

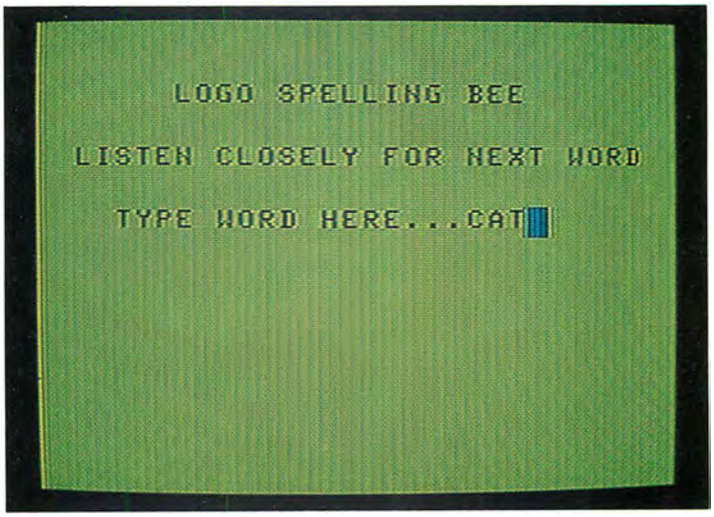




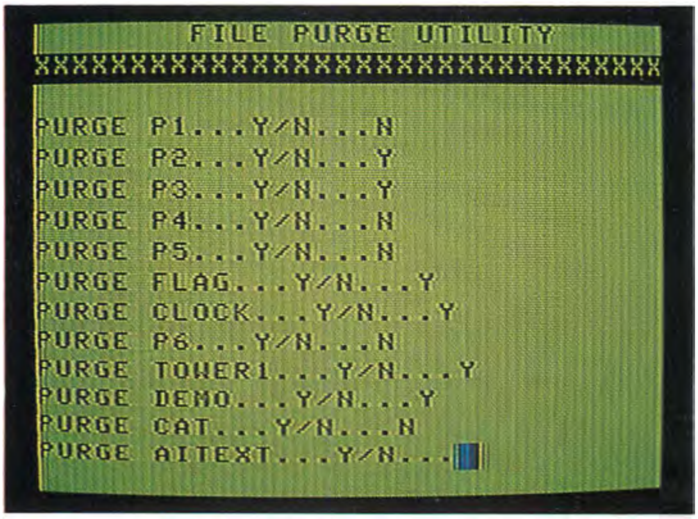




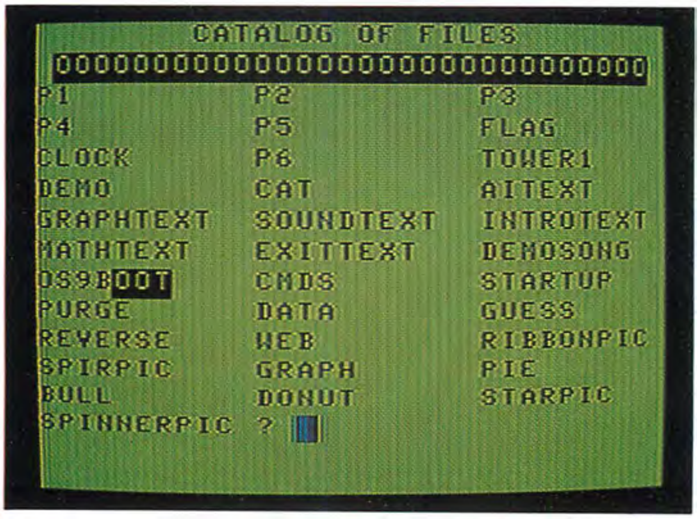
```
          MATH CALCULATOR
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


NUMBERS TO CALCULATE...144 12
TYPE THE SYMBOL FOR THE
OPERATION (-+*↑/)?...

THE ANSWER IS...12
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

?
```



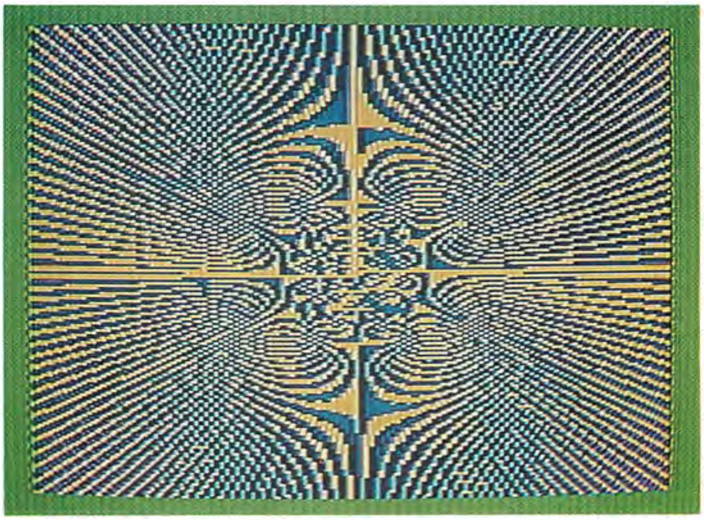
```
        LOGO SPELLING BEE

  LISTEN CLOSELY FOR NEXT WORD

     TYPE WORD HERE...CAT
```

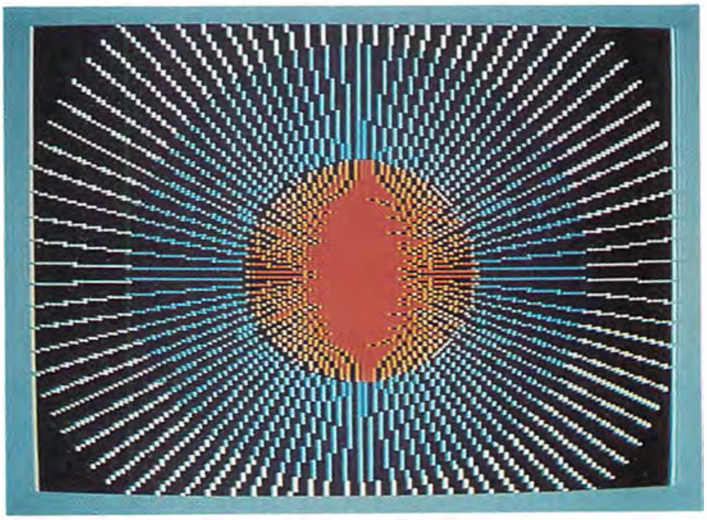


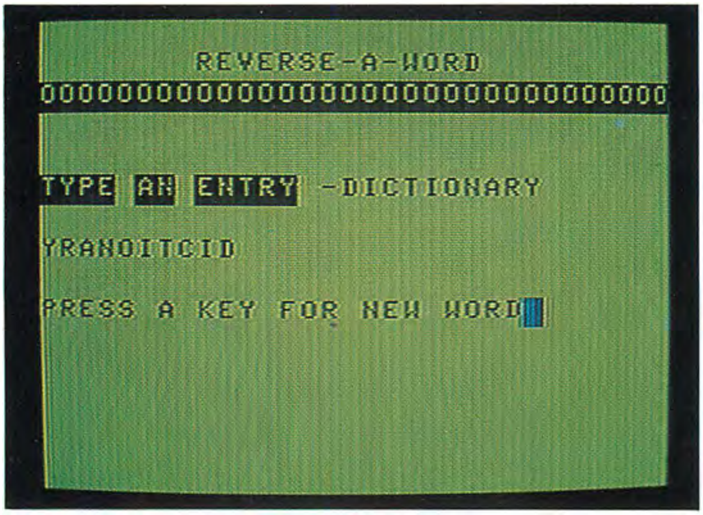
```
         FILE PURGE UTILITY
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

PURGE P1...Y/N...N
PURGE P2...Y/N...Y
PURGE P3...Y/N...Y
PURGE P4...Y/N...N
PURGE P5...Y/N...N
PURGE FLAG...Y/N...Y
PURGE CLOCK...Y/N...Y
PURGE P6...Y/N...N
PURGE TOWER1...Y/N...Y
PURGE DEMO...Y/N...Y
PURGE CAT...Y/N...N
PURGE AITEXT...Y/N...
```



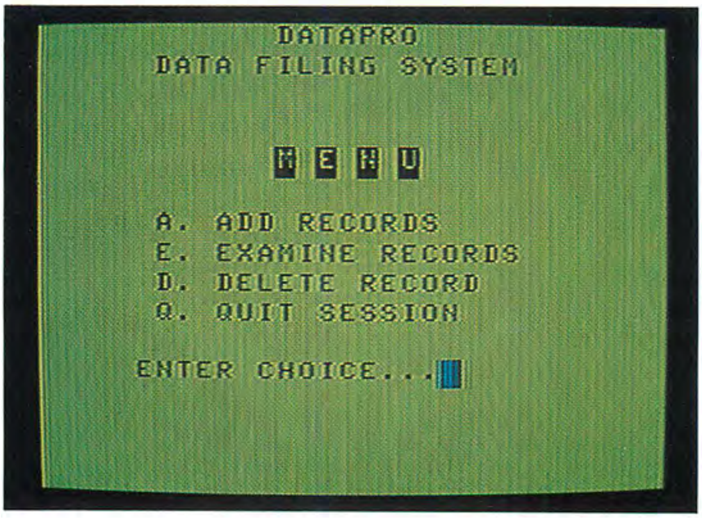
```
        CATALOG OF FILES
 00000000000000000000000000000000
P1          P2          P3
P4          P5          FLAG
CLOCK       P6          TOWER1
DEMO        CAT         AITEXT
GRAPHTEXT   SOUNDTEXT   INTROTEXT
MATHTEXT    EXITTEXT    DEMOSONG
OS9BOOT     CMDS        STARTUP
PURGE       DATA        GUESS
REVERSE     WEB         RIBBONPIC
SPIRPIC     GRAPH       PIE
BULL        DONUT       STARPIC
SPINNERPIC  ?
```



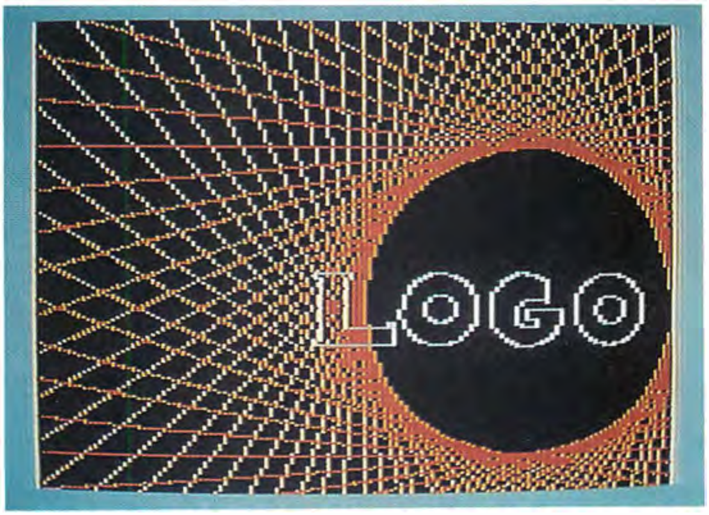


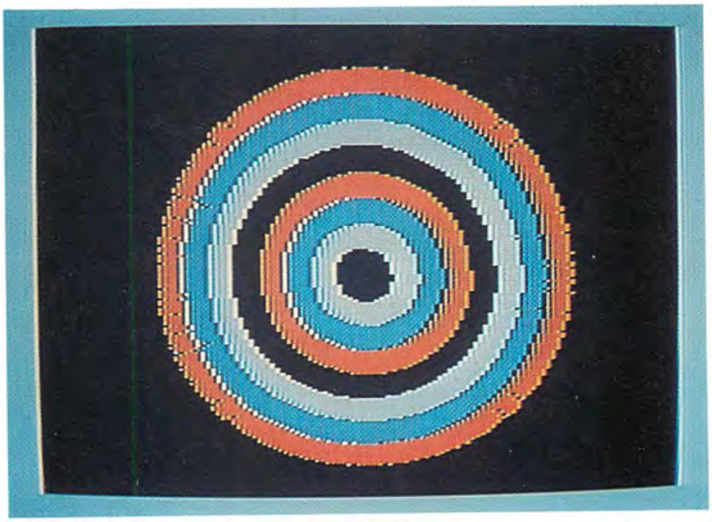


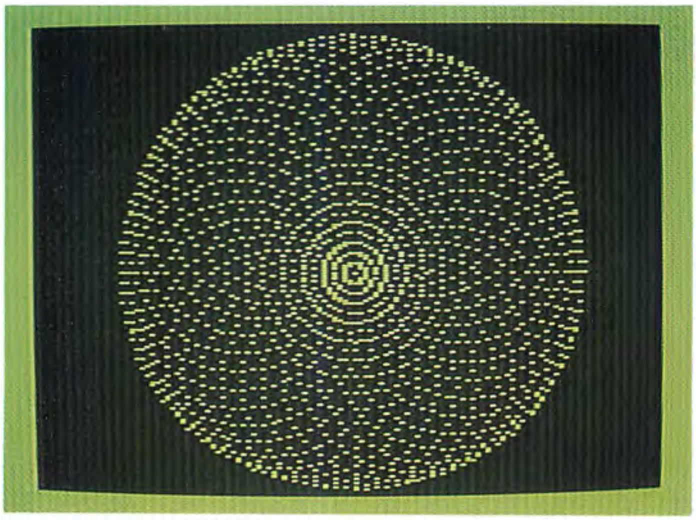
```
         REVERSE-A-WORD
0000000000000000000000000000000000

TYPE AN ENTRY -DICTIONARY
YRANOITCID
PRESS A KEY FOR NEW WORD
```

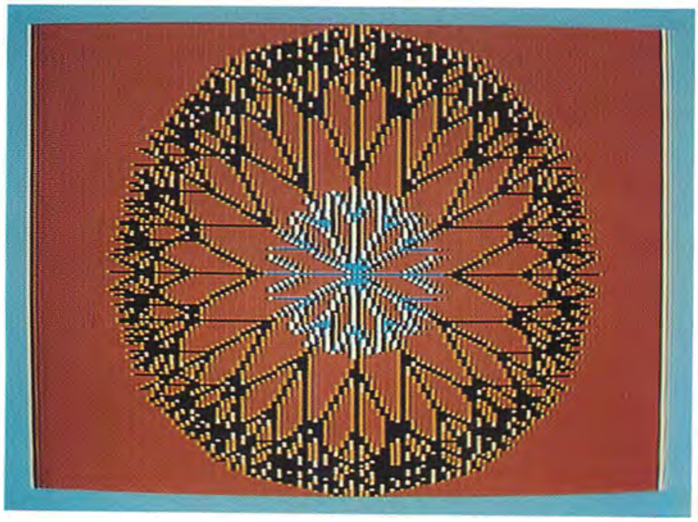


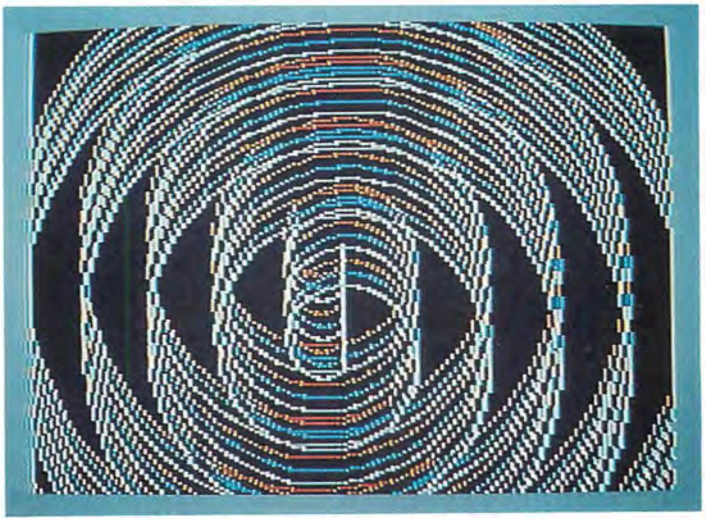
```
        DATAPRO
  DATA FILING SYSTEM

        M E N U

  A. ADD RECORDS
  E. EXAMINE RECORDS
  D. DELETE RECORD
  Q. QUIT SESSION

 ENTER CHOICE...
```

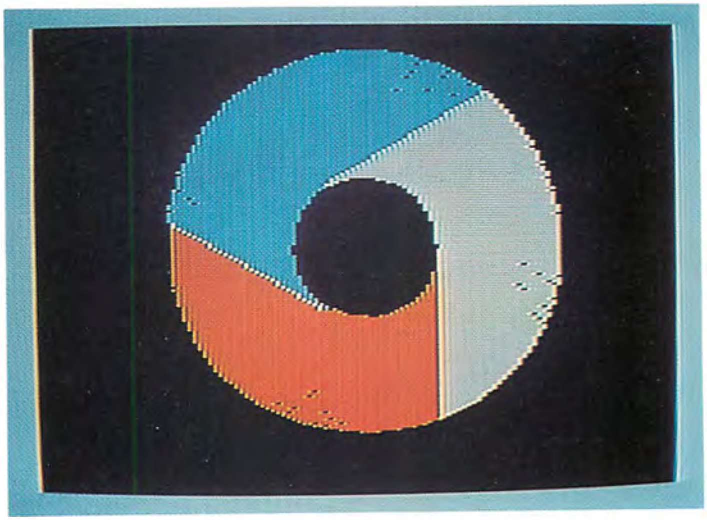

LOGO

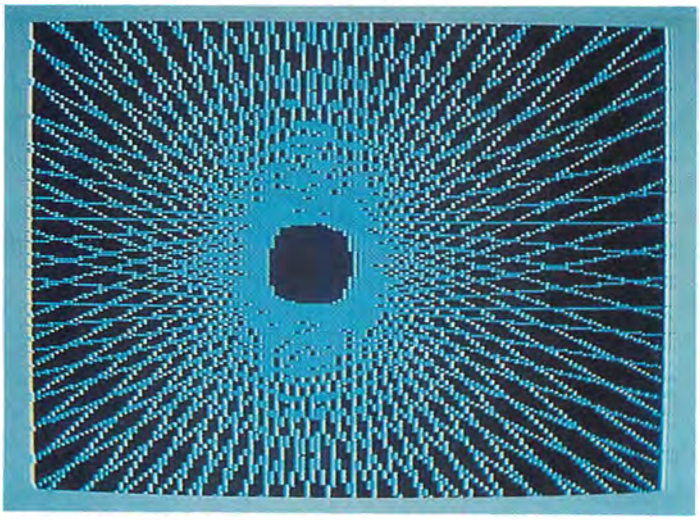


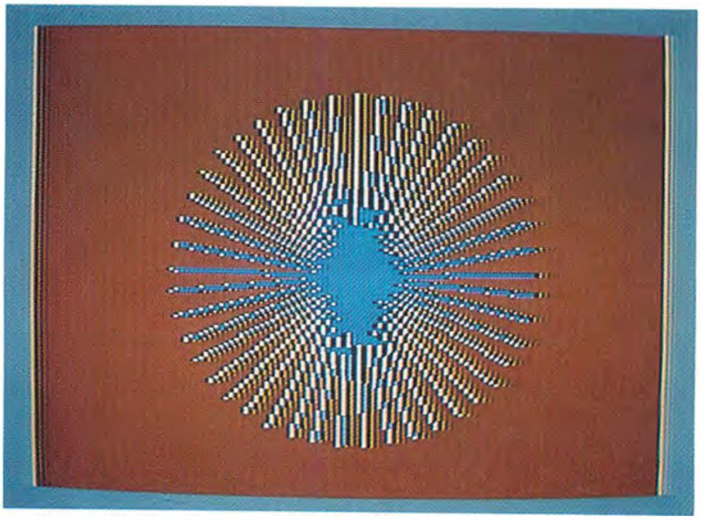


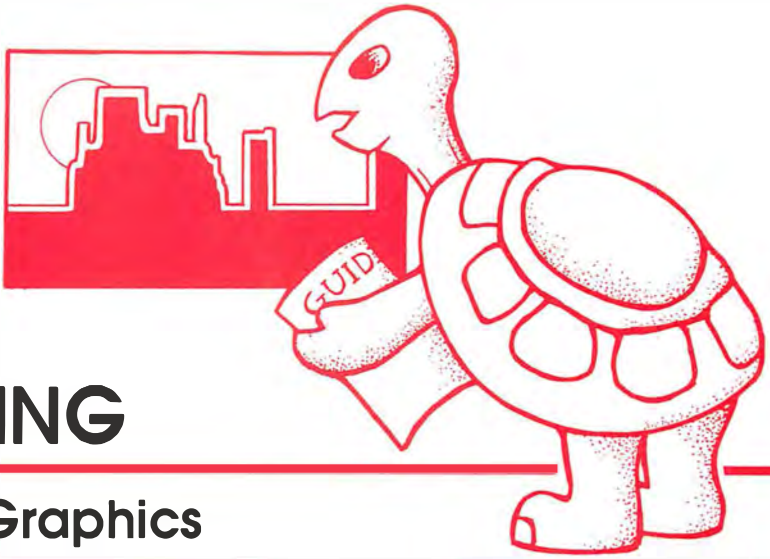












1 SIGHTSEEING

Introductory Graphics

Section 1	The Keyboard: special keyboard keys and functions.
Section 2	LOGO and the Primitives: FORWARD, BACK, RIGHT, LEFT, and HOME.
Section 3	Proceeding On: multiple commands to instruct your Turtle in a task.
Section 4	Abbreviations and Directions: short forms of primitives and the graphics screen layout.
Section 5	Play it Again: how to use the REPEAT primitive to execute a command any number of times.
Section 6	Bye: how to exit the LOGO program when you finish.

Section 1
The Keyboard

Because the keyboard on your Color Computer is similar to the keyboard on a conventional typewriter, you are probably already familiar with most of its functions. You need to be aware of several differences, however. The following chart describes the functions of several keys that are important to D.L. LOGO.

Key or Keys	Function
SHIFT	Several keys on your keyboard have 2 characters. For example, all the number keys have alternate punctuation and character symbols. To produce the top character on any key showing 2 characters, hold down SHIFT and press the desired character key.
ENTER	Completes primitive or command entries. To execute a primitive or command line, press ENTER.
BREAK	Stops execution of a procedure. Note: there may be a delay while the current command completes its operation.
→	Completes primitive or command entries in the immediate, or single command, mode. In the edit mode, → moves the cursor 4 spaces to the right. See Chapter 15 for information on keyboard functions in the edit mode.

. . . . About the Keyboard

The result generated by several of your computer keys may not be familiar to you. Now is a good time to experiment with these keys. A number of the keypress sequences are required frequently in D.L. LOGO, such as CTRL (, CTRL), and SHIFT BREAK. There is no danger in experimenting with any of the keys, you cannot damage your computer or D.L. LOGO.

. . . . About Keyboard Entries

In this manual, and on the video screen, a question mark (?) precedes the data you enter from the keyboard. Characters not preceded by a question mark are computer output in response to your keyboard entry.

. . . . About Active Keys

You have noticed that certain keys cause an immediate action when pressed. Such keys are called active keys. *The* CTRL BREAK *keypress sequence is an often used example of active keys. You have a number of such active keys when working in D.L LOGO's edit mode. Active keys do not require the* ENTER *key to complete their function.*

Key or Keys	**Function**
↑ ↓	Completes primitive or command entries in the immediate, or single command, mode.
←	Moves the cursor 1 space to the left. If you make a typing mistake, you can use the left arrow to move backward to the mistake, and then type the correct entry.
CTRL 0	Toggles uppercase and lowercase. When you first turn on your computer, the keyboard types in uppercase letters. Pressing CTRL 0 causes the computer to produce lowercase (green on black) letters. To return to uppercase letters, press CTRL 0 again. **Note: D.L. LOGO requires uppercase characters in primitive names.**
CTRL (and CTRL .)	Produces the corresponding square bracket [or].
CTRL A	Produces a repetition of the last keyboard entry if you are in the immediate (single command) mode or the graphics mode. You can use this feature to reexecute a command. In other words, you don't have to type the command again.
CTRL 7	Produces an up arrow (↑). The up arrow in D.L. LOGO is the arithmetic power symbol. For example, to produce 8^2, type: **8** CTRL 7 **2**.

Key or Keys	Function
SHIFT BREAK CTRL C	Changes screen modes. The 3 modes are the immediate (single command) mode, the FULLSCREEN graphics mode, and the SPLITSCREEN graphics mode. You can change modes even while a LOGO program is running.
CTRL /	Produces a backward slash (\). When this symbol is used before certain special characters in D.L. LOGO, those characters are treated in the same manner as normal keyboard characters.
CTRL 3	Produces a reverse (green on black) up arrow symbol.
CTRL =	Produces a left arrow symbol (←).
CTRL ,	Produces a reverse (green on black) left bracket symbol ([).
CTRL .	Produces a reverse (green on black) right bracket symbol (]).
CLEAR	Some Color Computers use CLEAR rather than CTRL. If your computer does not have CTRL, use CLEAR.

. . . . About Modes

You can think of modes as rooms. For instance, you have a kitchen where you prepare food and a bedroom where you sleep. In D.L. LOGO you have 3 modes: (1) the edit mode in which you write and edit procedures (2) the immediate, or single command, mode in which you view previous entries and issue housekeeping commands to accomplish such tasks as erasing procedures and (3) the graphics mode in which you see the computer execute graphics commands.

Section 2
Logo and the Primitives

A *primitive* is a built-in instruction to LOGO that causes it to perform some action. This manual refers to a primitive that is joined with its arguments, or parameters, as a *command*. For instance, FORWARD is a primitive; FORWARD 50 is a command. A group of commands is called a *procedure*.

. . . . About Syntax

Logo commands require a space between primitives and their arguments. LOGO accepts FORWARD 50 as a command, but FORWARD50 produces an error.

When you first start LOGO, you are in the immediate,or single command mode, and can begin instructing your Turtle. To do so, type:

```
? CLEARSCREEN [ENTER]
```

You are now in the graphics mode and can type primitives that cause immediate visual results.

You can return to the immediate mode in 2 ways. From the graphics mode, you can type **TEXTSCREEN** [ENTER], or you can press [SHIFT] and [BREAK] simultaneously until the immediate mode screen reappears.

. . . . About Steps

When the manual refers to steps, it refers to the distance your Turtle moves in units of 1. For instance, the distance across your screen from side to side is 256 steps. The distance from top to bottom is 192 steps. The command FORWARD 60 directs the Turtle to move forward 60 steps.

Forward

To see a graphics primitive in operation, type:

```
? FORWARD 30 [ENTER]
```

Your Turtle, which initially resides in the middle of the screen, moves up 30 steps, leaving a trail behind it. This line is *Turtle Graphics*.

Enter all LOGO commands in a manner similar to the FORWARD command. Type a command or command line, and press [ENTER].

Back

Try another command, this time, the BACK primitive. Type:

```
? BACK 30 [ENTER]
```

Be sure to press [ENTER] after typing this command. Your Turtle then returns home to the center of the screen. It draws a path each time, but, because it travels on the same path, only 1 line appears.

Turn Right

A Turtle with only 2 directions is rather limited. To provide more variety, type:

```
? RIGHT 90 [ENTER]
```

Watch the screen to see the Turtle spin 90 degrees. It now points to the right. To draw a line at right angles to the first, type:

```
? FORWARD 30 [ENTER]
```

Turn Left

Turn the Turtle left in the same manner as you turn it right. For example, to cause the Turtle to turn left a full 180 degrees and race forward 60 steps, type:

```
? LEFT 180 FORWARD 60 [ENTER]
```

Use BACK to accomplish almost the same maneuver. Type:

```
? BACK 60 [ENTER]
```

*. . . . **About Commands***

Issuing commands in LOGO is very similar to issuing commands in English. This feature makes LOGO easy to learn. You type FORWARD 50, and Turtle moves ahead 50 steps. You type RIGHT 90, and Turtle turns right 90 degrees. In fact, all of LOGO's primitives relate directly to English words.

*. . . . **About Errors***

If you make a mistake in commands you execute, LOGO provides an error message *to help you see what you have done wrong. For example, if you mistype FORWARD as FORWAD, the error message is* ***UNDEFINED PROCEDURE****. Because LOGO doesn't recognize FORWAD as a primitive, it thinks FORWAD is a procedure name. When it can't find a procedure defined as FORWAD, it tells you this.*

The Turtle moves back to the right end of the line without making a turn. However, it is still pointing left.

Give turn commands to your Turtle in degrees. Use 360 degrees to cause the Turtle to spin in a complete circle (for example: RIGHT 360). Turning the Turtle from an up direction to a left direction requires a LEFT turn of 90 degrees. An about-face is 180 degrees. Your Turtle can even turn fractions of 1 degree but, unless it does so repeatedly, you do not notice the change.

. . . . *About Directions*

This manual refers to the following directions throughout. Up *indicates the direction toward the top of the screen.* Down *indicates the direction toward the bottom of the screen.* Right *indicates the direction toward your right as you face the screen.* Left *indicates the direction toward your left as you face the screen.* Home position *indicates that the Turtle is in the center of the screen pointing toward the top of the screen.*

And Home

The HOME primitive brings your Turtle to the center of the screen from any location. To see the effect of this command, type:

```
? HOME [ENTER]
```

Not only does the Turtle come home, but it returns to its original position pointing toward the top of the screen.

. . . . *About the Turtle*

D.L. Logo's Turtle is represented by a box-shaped figure, with one end pointed. The direction this tip points is called the heading. *Changing the heading causes the Turtle to point in the indicated direction. The Turtle always draws in the direction indicated by the heading until it encounters a command that changes that heading.*

To complete a cross on the screen, type:

```
? BACK 30 [ENTER]
```

Section 3
Proceeding

A procedure is a method you use to accomplish a particular task. It can be as simple as 1 command or as complex as 100 or more commands. Keeping procedures as simple as possible works best. Later chapters teach you to combine procedures to accomplish large tasks.

If you want to use procedures more than 1 time, either retype and reenter the procedure commands or use CTRL A as described in Section 1. In Chapter 3, you learn how to write procedures in LOGO's workspace that you can use as many times as you wish. In effect, these procedures become primitives, and you manage them in exactly the same way. Keep this in mind as you work through the manual. By creating and saving procedures, you write your own customized computer language.

To write your first procedure, type:

```
? LEFT 90 FORWARD 30 RIGHT 90
FORWARD 60 RIGHT 90 FORWARD 60
RIGHT 90 FORWARD 60 RIGHT 90
FORWARD 30 ENTER
```

If LOGO displays an error message, carefully type the command again, and be sure it is exactly as above. Remember, if you make a mistake while typing a command, you can use ← to back up to the mistake and type over the incorrect character.

Following the typed instructions, the Turtle makes its first trip around the block, enclosing the previously created cross in a box.

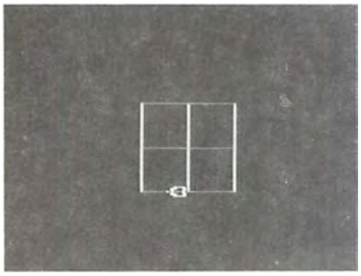
Later, you must give every procedure a name. For instance, you can name the preceding procedure BLOCK. (See Chapter 3 to learn how to name a procedure.)

. . . . *About Wraparound*

Many commands you type are longer than the 32 column width of your display screen. When the command line reaches the right hand edge of the screen, D.L. LOGO automatically causes a linefeed. *(The cursor drops one line and returns to the left edge of the screen.) Do not be concerned with this, but continue typing. D.L. LOGO executes command lines properly, whether they occupy 1 or several lines. D.L. LOGO has no trouble recognizing primitives or other words which are split between two lines. To make program lines easier to read, the manual splits long lines between words.*

. . . . *About Listings*

The procedure listing on this page must be typed as one, long, continuous line. Although the listing is shown as 4 separate lines, this is due to the size limitations of the page. Do not press ENTER *until the entire procedure is typed.*



Making It Clear

To proceed, clear your screen and type:

```
? CLEARSCREEN [ENTER]
```

The display graphics disappear, and the Turtle returns to its home position. To clear the text on the screen, type:

```
? CLEARTEXT [ENTER]
```

Whenever the graphics or text screens become cluttered, use CLEARSCREEN and CLEARTEXT to provide a fresh slate.

Section 4 Abbreviations and Directions

By now, you are probably tired of typing long Turtle primitives. There is often a better way. Many LOGO primitives have acceptable abbreviations. For instance, the abbreviation for FORWARD is FD. Try typing:

```
? FD 8Ø [ENTER]
```

Your Turtle scoots up 8Ø steps. It likes the FD primitive just as much as the longer FORWARD primitive.

Most of the primitives introduced earlier have abbreviations:

FORWARD	FD	LEFT	LT
BACK	BK	CLEARSCREEN	CS
RIGHT	RT		

There are no abbreviations for the HOME and CLEAR-TEXT primitives.

You encounter other primitive abbreviations as you go through the manual.

Making Your Move

The Turtle's HOME position in the center of the screen has the coordinates Ø,Ø. To move the Turtle from this position toward the top of the screen, you instruct it to move forward any number of steps, for instance, FD 4Ø. To move to the right, instruct the Turtle to turn right before moving forward. Moving down and moving left requires negative numbers. Therefore, if the Turtle moves 1Ø steps to the left and 1Ø steps down, its location is −1Ø, −1Ø.

*. . . . **About Arguments***

The term argument *is used extensively to refer to data or values upon which a primitive acts. For instance, in the command RIGHT 9Ø,* RIGHT *is the command and 9Ø is the argument. Some primitives do not require any arguments, such as CLEARSCREEN, while other primitives can accept 1 or more arguments.*

*If you type FORWARD/1, the error message is **SYNTAX ERROR**. LOGO doesn't recognize FORWARD/1 as a command, procedure, or any other function in its repertoire. It tells you that you have made a mistake in typing. See Chapter 12 for a list of error messages.*

. . . . *About Graphics*

This manual uses the word graphics *to refer to the pattern your Turtle draws on the display screen. Your computer is capable of 3 display screens: a text display, a graphics display, and a combined text-graphics display. A text display is a display of characters, such as letters and numbers. A graphics display is any display other than a text display. A combined text-graphics display consists of a specified number of text lines at the bottom of a graphics display. You learn how to set the number of graphics text lines in the next chapter.*

Readers familiar with geometry notice that the Turtle uses the Cartesian coordinate grid system. It may be helpful to picture the grid as a city block system. North and East are represented by positive numbers; South and West are represented by negative numbers.

From home position, you move your Turtle to the corner of 10 and 10 by commanding it to move forward 10 steps, turn right, and again move forward 10 steps (FD 10 RT 90 FD 10). To put the Turtle on the corner of 10 and −10, send it back 10 and right 10 (BK 10 RT 90 FD 10). A later chapter teaches you how to *teleport* the Turtle to any position on the grid system without leaving a trail behind.

To explore the bounds of your graphics screen, type the following commands.

```
? HOME ENTER
? FD 96 ENTER
? HOME ENTER
? BK 95 ENTER
? HOME ENTER
? RT 90 ENTER
? FD 127 ENTER
? HOME ENTER
? LT 90 ENTER
? FD 128 ENTER
```

As you see, the graphics screen is 192 units high by 256 units wide. The coordinates range: top to bottom = 96 to −95; left side to right side = −128 to 127.

Section 5
Play it Again

Turtles have perfect memories; at least, your LOGO Turtle does. Once you teach it a task, Turtle can repeat it perfectly. You only need to tell Turtle what the task is and how many times to do it.

The REPEAT primitive for multiple executions of a command, or series of commands, is a powerful LOGO primitive.

Before using REPEAT, you need to recall how to create square brackets ([ ]). To type these characters, hold down [ ] (if your computer does not have [CTRL], use [CLEAR]) and press either the left or right parenthesis to generate either the left or right square bracket.

Now, to test the REPEAT primitive, type:

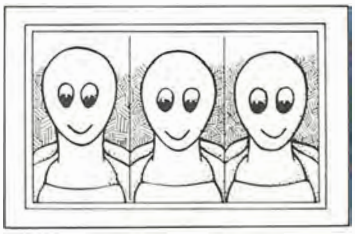
```
? CS REPEAT 4 [FD 30 RT 90] [ENTER]
```

Turtle clears the screen and then creates a box by drawing a line and making a right turn, 4 times. To get a preview of the real power of the REPEAT primitive, type:

```
? CS [ENTER]
? REPEAT 9 [RT 40 REPEAT 4 [FD
30 RT 90]] [ENTER]
```

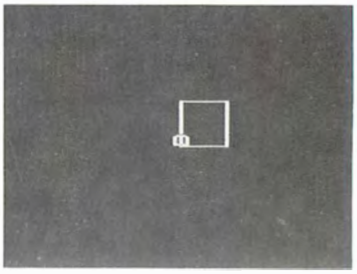
This produces a design that looks like the accompanying picture. If not, type the command again, and be sure it matches the example. The REPEAT primitive is the key to creating many procedures.

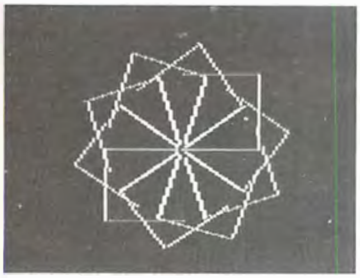
To see how Turtle accomplishes its trick, study each of the series of commands from right to left.



. . . . About Negative Numbers

D.L. Logo can handle negative numbers as arguments as well as it can handle positive numbers. When used with a command, such as FORWARD, a negative number has a reverse effect. For instance, the command FD −50 causes the Turtle to go backward 50 steps. The command RT −90 causes the Turtle to turn left 90 degrees. This can be a handy feature when using the results of calculations as arguments in commands.





FD 30 RT 90 sends the Turtle forward 3 steps and turns it right 90 degrees. If you're not sure how this works, clear your screen and execute this portion of the command again.

REPEAT 4 repeats the *draw line, turn right* activity 4 times, creating a box. The Turtle returns to home position after completing the last side of the box.

RT 40 turns the Turtle 40 degrees from home position.

REPEAT 9 repeats the procedure of drawing a square and turning 40 degrees 9 times. This completes a full 360 degree turn (9 times 40), with Turtle drawing a box at each turn. Turtle finishes in home position.

Section 6
Bye

Later chapters include information about saving and loading procedures, programs, and files. As you do not yet need to save your procedures on a diskette, exit LOGO by typing:

```
? BYE [ENTER]
```

You are now in the OS-9 operating system. When you finish using the computer, take any diskettes out of your drive(s), turn off the computer, and then turn off all other equipment.

Chapter Ends

This page completes the first chapter of instructions. The next few pages consist of a chapter summary and a suggested project. The summary is both a review of the chapter and a reference for the future. Try the suggested project to demonstrate how well you understand the chapter. If you have trouble, check the summary for help. If you need further help, review the chapter section that deals with the primitive with which you are having trouble. The last page of the chapter provides 1 possible solution to the suggested project. Your solution may be different and better. Most of the chapters in this manual include similar summaries and projects.

. . . About Chapter References

Each chapter in this manual ends with a chapter summary that includes a quick reference to the primitives introduced in that chapter. You may find it helpful to use this reference often as you learn about new procedures.

Chapter Summary

PRIMITIVE	Abbrev.	Purpose
FORWARD	FD	Moves Turtle forward a specified number of steps.
BACK	BK	Moves Turtle backward a specified number of steps.
RIGHT	RT	Turns Turtle right a specified number of degrees in the range 0 to 360.
LEFT	LT	Turns Turtle left a specified number of degrees in the range 0 to 360.
HOME	–	Brings Turtle back to home position (to the center of the screen pointed up).
CLEARSCREEN	CS	Clears the graphics screens and returns Turtle to the home position.
CLEARTEXT	–	Clears the text screens.
REPEAT	–	Causes Turtle to repeat commands a specified number of times.
BYE	–	Causes D.L. LOGO to exit to OS-9.

Turtle Facts

- You must:

  Separate primitives from parameters with a space.

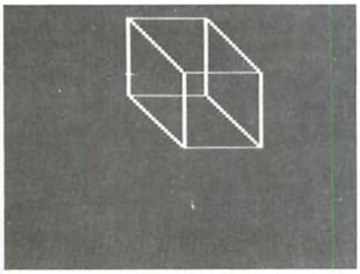
  Separate commands from other commands with a space.

  Execute commands by pressing ENTER.

- Turn the Turtle by specifying degrees; 360 is a full turn.
- Control the Turtle with either full or abbreviated primitive names.
- Create square brackets ([ ]) by holding down CTRL (if you computer does not have CTRL, use CLEAR) and pressing either the right or left parenthesis.
- The graphics screen is 256 steps, side to side, and 192 steps, top to bottom. The coordinates for the center position of the screen grid are 0,0.

Suggested Project

Write the commands to draw a cube as shown in the accompanying picture. You can write the procedure as 1 command line or as a series of separate commands. If you have trouble, review the preceding summary. If you still have trouble, review the section of the chapter that deals with the primitive that is causing you problems. See the next page for a possible solution.



Suggested Project Solution

```
REPEAT 4 [FD 50 RT 90]
LT 45 FD 50 RT 45
REPEAT 4 [FD 50 RT 90]
FD 50 RT 135 FD 50 LT 45
FD 50 LT 135 FD 50
LT 135 FD 50 LT 45 FD 50 ENTER
```

2 TECHNICOLOR TURTLE

Discovering Turtle's Many Colors

Section 1 Turtle Shows Its True Colors: selecting pen color.

Section 2 Setting Background Colors: selecting screen colors.

Section 3 Calculating Colors: using PENCOLOR and BACKGROUNDCOLOR in calculations.

Section 4 Housekeeping: clearing the screen and defining screen formats.

Section 5 About the Textscreen: writing commands and procedures in the immediate mode.

The sample procedures in this chapter assume that LOGO is in the default background color of 12 and pen color of 3. If you have changed these settings, you may wish to reset the background and pen colors to match the manual. To do so, type the following command while in either the immediate mode or the graphics mode.

```
SETBG 12 SETPC 3 [ENTER]
```

Section 1 Turtle Shows Its True Colors

Until now, you saw your Turtle drawing only buff lines on a black screen. Many other options are available.

To tell the Turtle to use another pen color, issue the SETPENCOLOR primitive. You can abbreviate this primitive to SETPC. To switch pens from 3 to 1, type:

```
? SETPC 1 ENTER
```

The Turtle puts its buff pen in its pocket and replaces it with a cyan pen. To be sure Turtle has done this, draw a line in the new color by typing:

```
? FD 3Ø ENTER
```

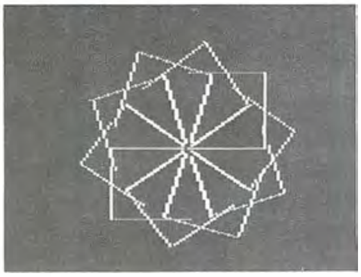
You can select 1 of 4 pen colors (Ø-3). With each of the 16 possible background colors, you have an option of 4 pen colors. To see all of the current options, type:

```
? CS ENTER
? SETPC Ø ENTER
? REPEAT 4 [FD 5Ø RT 9Ø SETPC
PC+1] ENTER
```

Your screen now displays a 3-sided box, with each side a different color. Why does the box have only 3 sides when the command asks for 4 lines? Color Ø does not show because SETPC Ø sets the pen color to the same color as the screen. On the unseen fourth side of the box, the Turtle draws a black line on a black screen. The 3 colors showing are green, red, and buff. Other combinations of foreground and background colors cause different sides of the box to be invisible, depending on which side of the box matches the background color.

. . . . *About Screen Colors*

The colors on your television screen may not always seem to match the colors described in this manual. The color shades depend on your television and how you have set its tint, color, contrast, and brightness.



Although this shape is simple and easy to construct, you can create a more complex and attractive design by adding more commands. Try this:

```
? CS ENTER
? REPEAT 10 [RT 36 SETPC 0
REPEAT 4 [FD 50 RT 90 SETPC
PC+1]] ENTER
```

Experiment with other values as well.

Section 2 Background Colors

By changing the screen background color, you can select other pen colors. Both pen and background have a total of 16 colors. You select the background colors with the primitive SETBACKGROUND or SETBG. To select an orange screen, type:

```
    ? SETBG 7 CS
```

The portion of the screen reserved for graphics changes to orange. To see the possible pen colors, reenter the previous command that draws a square with colored sides.

A chart showing all possible background and pen colors follows.

Background and Pen colors

Background		Pen Color			
		0	1	2	3
0	green	green	yellow	blue	red
1	yellow	green	yellow	blue	red
2	blue	green	yellow	blue	red
3	red	green	yellow	blue	red
4	buff	buff	cyan	magenta	orange
5	cyan	buff	cyan	magenta	orange
6	magenta	buff	cyan	magenta	orange
7	orange	buff	cyan	magenta	orange
8	black	black	dark green	medium green	light green
9	dark green	black	dark green	medium green	light green
10	medium green	black	dark green	medium green	light green
11	light green	black	dark green	medium green	light green
12	black	black	green*	red*	buff
13	green*	black	green*	red*	buff
14	red*	black	green*	red*	buff
15	buff	black	green*	red*	buff

*Red and green may be reversed.

Section 3 Calculating Colors

The design created at the end of Section 1 used an unfamiliar command. The command SETPC PC+1 is possible because the primitive PC always contains the current pencolor value.

For example, if you type **SETPC 1** to set the pen color to 1, typing **PC** displays *1* on the screen. The command SETPC PC+1 sets the pen color to 2, and the primitive PC displays 2.

The SETBACKGROUND primitive works in the same way, with BG containing the last value established by SETBG. To see the current background color, type **BG** ENTER. You can now add 1 to the background color by typing:

```
? SETBG BG+1 ENTER
```

Test this by again typing:

```
? BG ENTER
```

To see all the possible background and pen colors available in D.L. LOGO, enter the following program:

```
? SETBG Ø ENTER
? CS ENTER
? REPEAT 16 [REPEAT 4 [REPEAT
1Ø [FD 5Ø RT 9Ø FD 1 RT 9Ø FD
5Ø LT 9Ø FD 1 LT 9Ø ] SETPC
PC +1] CS SETPC Ø SETBG BG+1
CS] ENTER
```

If you make a mistake when typing long procedures, remember you can use CLEAR A to repeat the line. Then backup to the mistake, correct it, and retype the rest of the program.

*. . . . **About Color Settings***

Although there are only 4 pen colors (Ø-3), D.L. LOGO still lets you set the pencolor to 4 or higher. This is because the colors repeat if you exceed their normal settings. A pencolor of 4 is the same as a pencolor of Ø. A pencolor of 5 is the same as a pencolor of 1. The background colors behave in exactly the same way, and a background setting of 17 is the same as a background setting of Ø. This repetition will continue to a maximum value of 32767, after which D.L. LOGO will generate a NUMBER OUT OF RANGE error.

This procedure draws 4 solid boxes and shows all the possible pen colors on each of the 16 background colors. One of the 4 boxes is the same color as the background and is invisible.

Section 4 Housekeeping

Issuing the CS or CLEARSCREEN primitive introduced in Chapter 1 sends the Turtle back home. The CLEAN primitive also clears the screen but leaves the Turtle where it is. To test it, type and enter the following procedure:

```
? SETPC 3 ENTER
? REPEAT 4 [REPEAT 4 [FD 40 RT
90] FD 10 CLEAN] ENTER
```

This procedure draws a box in 4 different screen positions, as illustrated. Press CTRL A to reenter the same command. Then, use ← to back up the cursor and change the command by replacing the CLEAN primitive with the CS primitive:

```
? CS ENTER
? REPEAT 4 [REPEAT 4 [FD 40 RT
90] FD 10 CS] ENTER
```

As you can see, the results of the 2 procedures are quite different. The CS primitive always draws the box in the same position on the screen. CLEAN causes the box to appear to move, because the location where the Turtle stops in drawing a box becomes the starting point for the next box.

FULLSCREEN and SPLIT ... select the view

So far, the Turtle's activities occurred within the available display area. At times, however, you need to use the entire display screen. The FULLSCREEN primitive is specifically designed to allow this. To draw a triangle on the screen, type the following procedure:

```
? CS [ENTER]
? REPEAT 3 [RT 120 FD 85] [ENTER]
```

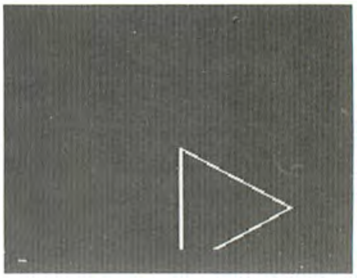
Part of the triangle you created appears cut off at the bottom. Cure the problem by typing **FULLSCREEN** [ENTER]. The text part of the screen now disappears, and the screen displays the full triangle.

When you start LOGO, it reserves 2 lines of text on the graphics screen. Issuing the FULLSCREEN primitive clears the text lines and reserves the full screen for graphics.

When this happens, the LOGO prompt no longer appears on the screen, and nothing appears when you type. LOGO is not ignoring your keystrokes; it is reserving the whole screen for the Turtle. To see what you are typing, use [SHIFT] [BREAK] to toggle to a split screen or to the text screen in the immediate mode. You can also return the graphics screen to the normal split screen mode by issuing the SPLITSCREEN primitive. To do so, type **SPLITSCREEN** [ENTER]. The same number of text lines you previously had reappears, showing the last 2 lines you typed. You can use the SPLITSCREEN primitive at the end of any procedure that sets the screen to FULLSCREEN. For example, typing:

```
? FULLSCREEN REPEAT 3 [RT 120
FD 85] [ENTER]
```

draws a triangle using the full screen. To restore text lines to the graphics screen, type **SPLITSCREEN**.



*. . . . **About Split***

Although you can set 15 lines of the graphics screen for text, this is normally impractical. You can do little with only 1/16th of the screen for graphics. However, as graphics commands operate behind text lines, you can use SPLITSCREEN to hide graphics and then reveal them with the FULLSCREEN primitive. Note that the use of the SETSPLIT primitive clears all graphics from the screen. To hide graphics in this manner, use SETSPLIT to set the proper number of text lines before you draw the graphics.

Split 15 Ways

The SETSPLIT primitive lets you set from 1 to 15 lines of text on the graphics display. To set 0 lines, use the FULLSCREEN primitive. For example, typing **SETSPLIT 10** reserves 10 lines of the graphics screen for text. LOGO can also tell you the condition of the graphics screen. When LOGO initializes, it has a split setting of 2 lines. You can determine this by typing **SPLIT** ENTER. The screen displays 2.

Chapter 2 Summary

Summary

PRIMITIVE	Abbrev.	Purpose
SETPENCOLOR	SETPC	Establishes the current pen color. There are 4 pen colors for each background color.
SETBACKGROUND	SETBG	Establishes the current background or screen color. There are 16 background colors.
PENCOLOR	PC	Displays the current pen color.
BACKGROUND	BG	Displays the current background color.
FULLSCREEN	–	Reserves the entire graphics screen for graphics and sets the number of graphics text lines to Ø.
CLEAN	–	Clears the graphics screen without moving current Turtle coordinates.

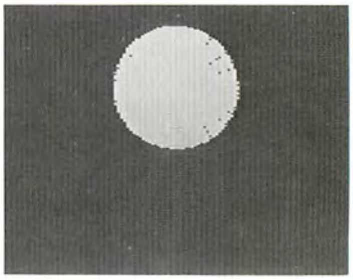
PRIMITIVE	Abbrev.	Purpose
SPLITSCREEN	–	Reserves text lines in the graphics screen mode.
SPLIT	–	Displays the current graphics screen split value.
SETSPLIT	–	Sets text lines in the graphics mode. SETSPLIT can be in the range 1 to 15.
TEXTSCREEN	–	Causes D.L. LOGO to return from the graphics mode to the immediate.

Turtle Facts

- When you initialize LOGO, the background color is set to 12, and the pencolor is set to 3.
- Four pen colors are available for each of the 16 background colors, one of which is always the same as the background color.
- To get the proper background color, issue a CLEARSCREEN (CS) primitive after setting the color.
- With some background settings, the area reserved for text is a different color from the background color.
- You can display several LOGO settings and use the settings in calculations. PENCOLOR (PC), BACKGROUND (BG), and SPLIT are some.
- CLEARSCREEN (CS) always returns the Turtle to its home position. CLEAN does not.

- When working in the FULLSCREEN graphics mode, the screen does not display the text you type.

- To view the full graphics screen, issue the FULL-SCREEN primitive or toggle screens with SHIFT BREAK.



Suggested Project

Write a procedure that draws a filled circle on the screen. Use the SETPC primitive to draw the circle 4 times with 4 different colors. Of course, 1 of the colors will be the background color.

For a greater challenge, expand the program to draw 4 circles in each of the 16 background colors.

Suggested Project Solution

To draw a single filled circle

```
REPEAT 18Ø [FD 8Ø RT 179] ENTER
```

To draw 4 circles

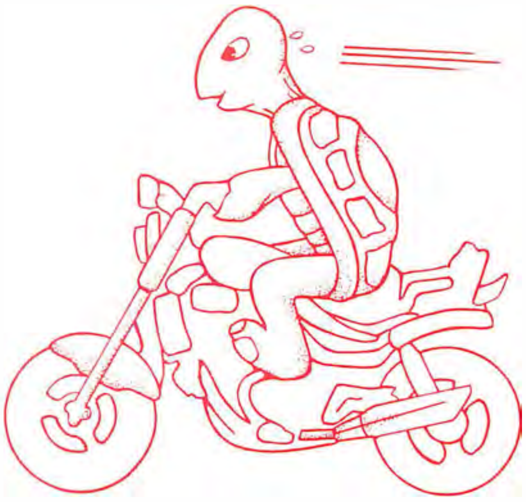
```
SETPC Ø ENTER
REPEAT 4 [REPEAT 18Ø [FD 8Ø
 RT 179] SETPC PC + 1] ENTER
```

Note that the first circle is the same color as the background color and does not show.

To draw circles in 16 background colors

```
SETPC Ø
SETBG Ø
REPEAT 16 [REPEAT 4 [REPEAT 18Ø
 [FD 8Ø RT 179] SETPC PC+1] SETPC
 Ø SETBG BG + 1 CS] ENTER
```

3 RUNNING ERRANDS



Procedures for Your Turtle

Section 1	Writing Procedures: constructing procedures for multiple use.
Section 2	About Editing: what to do when a procedure doesn't work or needs changing.
Section 3	Direct Wire: sending information directly to LOGO procedures from the keyboard.

This chapter assumes that you are beginning under the D.L. LOGO startup defaults. That is, the background color is 12 and the pen color is 3.

Section 1
Writing Procedures

Because of the way LOGO handles procedures, you can build complex programs from simple commands. The ability to link and nest procedures provides virtually unlimited potential to the programmer.

D.L. LOGO's edit mode lets you develop procedures to their full potential:

- You do not destroy procedures written in the edit mode when you execute them.
- Procedures can call other procedures to do a part of a particular task.
- You can save procedures on diskette to use later or as part of a procedure library.

Working in the Edit Mode

Enter the edit mode either by defining a procedure name or by typing **EDIT** from the immediate or graphics mode. To define a procedure name, type **TO *procedurename***. The word *procedurename* is used to represent the actual name you wish to give your new procedure. You can also enter the edit mode by typing **EDIT** ENTER. When you do this, you must press I (for insert) before you can begin writing a procedure.

Although writing a procedure in the edit mode is similar to writing it in the graphics or immediate mode, there are 2 main differences:

- Pressing ENTER does not execute a line.
- Each procedure must begin with a name (preceded by TO) and end with the primitive END.

. . . . About Procedure Theory

Logo is considered to be a procedural programming language. Procedures are small programs that collectively build a larger program. Using small procedures lets you break a complex program into a number of small manageable parts.

. . . . About Procedure Names

*You can give a procedure any name you wish as long as it doesn't begin with a numeral or contain any of the following symbols: ! " ( ) * = – ⟨ ⟩ /. Although D.L. LOGO accepts names that include other symbols, it eliminates the symbol and any characters that follow it when you save the procedure on diskette. If you save a procedure named TEST#1 and a second procedure named TEST#2, D.L. LOGO saves both procedures as TEST, and the latter procedure overwrites the former and destroys it.*

The First Procedure

Try an old procedure in the new edit mode. From the immediate mode, type **TO BOX** ENTER. The display clears, and the words TO BOX appear at the top of the screen. The blue cursor is located 1 line below this procedure name.

You are now in the edit mode and can complete the procedure. Begin by typing **REPEAT 4 [**. Remember that the square bracket is produced by pressing CTRL, together with the appropriate parenthesis key. When you complete a line, press ENTER before beginning a new line. Type the remainder of the program to match the following listing:

```
TO BOX
REPEAT 4 [ FD 40 RT 90]
END
```

When you enter the edit mode by defining a procedure (TO BOX), you are automatically in the insert mode; LOGO inserts the text you type into the procedure. In this case, the procedure was empty; therefore, what you insert is the procedure.

To return to the immediate mode, press BREAK twice: once to exit the insert mode, and once to exit the edit mode. From the immediate mode you can execute your new procedure. To do so, type **BOX** ENTER.

. . . . About Spaces

A procedure name cannot contain spaces. You can use other characters to divide words in names such as miles-gone, phone-list, *or* budget$spent.

Be sure that procedure names you select are not already defined as LOGO primitives.

. . . . About Lines

In the edit mode, you can put as many characters in a line as you wish. However, in the immediate mode, you cannot put more than 255 characters in a command line.

Building a Program

After you define and write a procedure, you can use it as part of a larger program. To see how this works, return to the edit mode by typing **EDIT** ENTER. Now add a new procedure. First, hold down SHIFT and press G to place the cursor after the BOX procedure. Now press I to enter the insert mode and press ENTER to leave a blank line between the BOX procedure and your new procedure. To create the second procedure, type:

```
TO ROTATE
REPEAT 36 [ BOX RT 10]
END
```

When the procedure is complete, press BREAK twice to exit the edit mode. Execute ROTATE by typing **ROTATE** ENTER. The Turtle draws the same box but, this time, it repeats the pattern 36 times, rotating 10 degrees between each box.

*. . . . **About Choosing Names***

Although procedure names can be as long as you like, short names that describe the procedures they represent save time and simplify programming. For example, use the name BOX or SQUARE for a procedure that draws a square shape. If you create a routine that draws several boxes, name it MOREBOX. Call a procedure that draws a circle CIR.

Remember that the word TO always precedes a procedure name when you are writing a procedure. When you execute the procedure, do not include TO.

. . . . About Order in the Workspace

It does not matter in what order procedures are arranged in LOGO's workspace. Procedures called by another procedure can be either before or after the calling procedure.

. . . About Procedure Format

A procedure consists of a heading (the procedure name preceded by the word TO), a body (a series of LOGO statements), and an END statement. The style in which you format procedures is a personal choice. You can write the REC procedure with the procedure statements all as 1 continuous line, if you wish. For clarity, this manual separates commands and uses indention. Because the manual breaks lines between words, rather than in words, the manual listings may look different from your screen displays. This format makes it easier for you to follow the logic and flow of procedures.

Section 2
About Editing

Another advantage to writing programs in the edit mode is D.L. LOGO's sophisticated editor. Before you consider more procedures, learn how the D.L. LOGO editor works. Once you learn how to insert and delete in the edit mode, writing and changing procedures is easier.

The Editing Options

Fortunately D.L. LOGO's editing features are easy to understand and use. You may, however, wish to learn only the fundamentals of the editor program now and then study it later when you have more experience with LOGO. This section shows you simple ways to delete, insert, and change text. All of D.L. LOGO's edit features are explained in Chapter 15.

To enter the edit mode from the immediate mode or the graphics mode, type **EDIT** ENTER. To exit the edit mode, press BREAK. Another way to enter the edit mode is to define a procedure by preceding the procedure name with the primitive TO, such as:

```
TO JOB ENTER
```

Editing Sample Session

To create a procedure for the sample editing session, enter the edit mode from the immediate or graphics mode by typing **TO REC** ENTER. Now, type the following procedure. End each line by pressing ENTER.

```
TO REC
FD 40 RT 90
FD 5 RT 90
FD 40 RT 190
END
```

To see what your program does, press BREAK twice and, in the immediate mode, type **CS REC** ENTER.

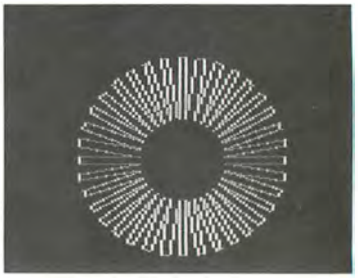
Your screen shows a 3-sided rectangular figure. To create something a little more impressive, reenter the edit mode by typing **EDIT** ⎵.

If your cursor is not already below the previously entered procedure, move it to the end of the listing by pressing SHIFT G. To begin a new procedure and enter the insert mode, press I ENTER.

Type the second procedure:

```
TO SPIN
SETSPLIT 1
PU FD 20 PD
REPEAT 36 [REC]
END
```

This completes the sample program. Before making any changes, execute the program. Type BREAK BREAK **SPIN** ENTER.



. . . . About PU and PD

The PU and PD primitives used in this procedure are abbreviations of the PENUP and PENDOWN commands. You learn about these commands later.

A Change of Scenery

Now, change the program so that it executes in different ways. For example, try alternating the color of the design. To do so, enter the edit mode by typing **EDIT** ENTER.

Use the arrow keys to position the cursor at the beginning of the first line of the SPIN procedure. The underline (_) represents the blue cursor and, in this case, covers the initial S of SETSPLIT:

```
TO SPIN
_ETSPLIT 1
```

Press SHIFT → to move the cursor to the end of the line:

```
SETSPLIT 1_
```

. . . . About Procedure Lines

When referring to lines in a program, the manual begins counting at the procedure's name line (the primitive TO followed by the procedure name). Thus in the SPIN procedure, Line 1 is TO SPIN, *and Line 2 begins* SETPC 1.

. . . . About Spacing

Procedures co-existing in the D.L. LOGO workspace must be separated by at least 1 blank line.

. . . . About Vocabulary

Whenever you define a new procedure, the name you give the procedure becomes part of LOGO's vocabulary. Thus, the procedure you define can be used in exactly the same manner as a primitive name, as long as the procedure is in the workspace.

Press **I** and enter the new command:

```
    SETPC 1
```

The procedure should now look like this:

```
    TO SPIN
    SETSPLIT 1 SETPC 1
    PU FD 20 PD
    REPEAT 36 [REC]
    END
```

Press BREAK again, and then type **CS SPIN** ENTER. Your design appears in blue instead of buff.

Type **EDIT** to return to the edit mode. To erase the line you just entered and replace it, use the arrow keys to position the cursor on the new line. Type D D to delete the entire line

To replace the line, use the insert command. From the same cursor position, type:

```
    I SETSPLIT 1 SETPC 2 ENTER
    BREAK
```

Pressing BREAK eliminates the gap that the insert command opens. To see the design in a new color, exit the edit mode, and execute the program as before.

To create the same graphics design in a third color, enter the edit mode, and use the arrow keys to position the cursor over the number 2. Now type:

```
    R3
```

Use [→] to move the cursor and confirm the change. Execute the program, if you wish, and then reenter the edit mode.

To make the program neater, split the current line into 2 lines. Move the cursor to the space between 1 and SETSPLIT 1:

```
TO SPIN
SETPC 1_SETSPLIT 3
PU FD 20 PD
REPEAT 36 [REC]
END
```

Replace the space with a carriage return by pressing [R], and then press [ENTER].

You can also change or replace a block of text. Position the cursor on Line 2 of the REC procedure:

```
TO REC
_D 40 RT 90
```

Now press [SHIFT] [R]. You are in the change mode and can replace as many characters in the line as you wish. Type:

```
BK 30 [BREAK]
```

Pressing [BREAK] exits the change mode and lets you see the new line. Move the cursor to the beginning of Line 4:

```
TO REC
BK 30 RT 90
FD 5 RT 90
_D 40 RT 190
```

To enter the change mode and replace FD with BK, type:

```
[SHIFT] [R] BK [BREAK]
```

. . . . About the Workspace

D.L. LOGO's workspace is the part of your computer's memory that is allocated for the temporary storage of procedures. You can manipulate procedures and lines in the workspace much like moving and manipulating the files in a file drawer. Procedures, lines, and characters can be moved, changed, or replaced at will, using D.L. LOGO's editing commands.

Pressing SHIFT R causes any character you type to replace the character under the cursor. You remain in the replace mode until you press BREAK. If you continue to type characters beyond the current line or procedure, D.L. LOGO continues replacing characters into any subsequent lines or procedures.

If you like, exit the edit mode and try the new program.

Other Changes To Make

As you can see, changing a program is easy. Other suggestions for using the editing commands you have learned are:

- Have your turtle draw the design in 3 colors without having to stop and edit the program. Hint: add a REPEAT 3 line before SETPC and replace the SETPC 2 line with SETPC PC+1. Set the pencolor to 1 before you execute the program.
- Change the size of your design by changing the value of FD 5 in the REC procedure. Try several different values.
- Change the FD 40 values in the REC procedure.
- Try unequal values in the 2 FD 40 commands.

Looking Neat

To write procedures that are easy to read, indent lines using →. For example, to clarify a loop, indent all the lines associated with the loop. Use the SPIN procedure to illustrate this. First, add some new lines to make the procedure more complex. To add the new lines, position the cursor where you wish to insert a line, and press I. Type the new line, followed by ENTER BREAK. Insert the necessary lines to change SPIN as shown:

```
TO SPIN
SETSPLIT 1
PU FD 20 PD
SETPC 3
REPEAT 4 [
REPEAT 36 [REC]
SETPC PC - 1]
END
```

Now, position your cursor at the beginning of the second line:

```
_SETSPLIT 1
```

and type I → BREAK

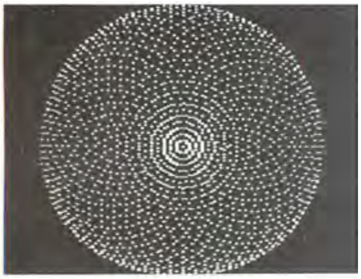
This indents the line 4 spaces. Do the same for the next 3 lines. On lines 6 and 7, press → twice to indent the lines 8 spaces. Leave the last line flush left. Your procedure now looks like this:

```
TO SPIN
    SETSPLIT 1 SETPC 3
    PU FD 20 PD
    SETPC 3
    REPEAT 4[
        REPEAT 36 [REC]
        SETPC PC - 1]
END
```

You can use → to indent lines as you write the procedure. The indention does not affect the procedure's operation in any way; instead, it makes the procedure easier to understand. This method of entering procedures is optional. The rest of this manual uses indention where applicable.

. . . . *About Using* ENTER

At times a line you are typing in the edit mode ends at the extreme right of your display screen. When this occurs, the cursor automatically drops 1 line and moves to the left of the screen (as though you had pressed ENTER*). Even though the cursor is at the beginning of a new line on the display, you must also press* ENTER *before beginning a new line or an error may result when the procedure is executed.*



. . . . About Comments

Comments are especially desirable in procedures or programs that you might want to adapt for other uses or that you wish to share with friends. Although the logic of a procedure seems clear at the moment, several months from now it can become obscure and difficult to interpret. Comments in programs you give to others can help them understand how the procedures might be adapted to their needs.

Although the editing procedures in this section let you make any changes to your programs or procedures, we suggest you also look over the editing procedures in Chapter 15. As you write larger and more complex procedures and programs, the many other editing features of D.L. LOGO are useful and timesaving.

Using Comments

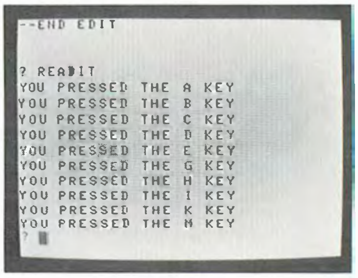
To create procedures that are easy to understand, it is often desirable to include comments at key points. You can include comments in a procedure that will be ignored by D.L. LOGO by preceding them with semicolons (;). Following is a procedure that is commented:

```
TO SWIRL
     SETPC 3
     CS HT PU
     REPEAT 36 [RT 10 ;MAKE A
      FULL CIRCLE
     REPEAT 72 [FD 4 RT 5 DOT
      XCOR
     YCOR]] ;MAKE A CIRCLE OF
      DOTS
END
```

Section 3
Direct Wire

LOGO has 2 primitives that cause a program to pause and wait for keyboard input. The READCHARACTER, or RC, primitive reads a single character from the keyboard. You need not press ENTER to register the character input. The following procedure demonstrates a READCHARACTER input:

```
TO READIT
      ST PD
      REPEAT 10 [
      MAKE "CHARACTER RC
      PRINT [YOU PRESSED THE]
       :CHARACTER [KEY]]
END
```



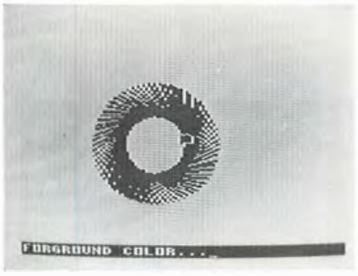
```
--END EDIT

? READIT
YOU PRESSED THE A KEY
YOU PRESSED THE B KEY
YOU PRESSED THE C KEY
YOU PRESSED THE D KEY
YOU PRESSED THE E KEY
YOU PRESSED THE G KEY
YOU PRESSED THE H KEY
YOU PRESSED THE I KEY
YOU PRESSED THE K KEY
YOU PRESSED THE M KEY
? █
```

To execute the procedure, type **READIT** ENTER from the immediate mode. You type a total of 10 keys, and each time the procedure tells you which key you pressed.

To see how you can use the RC primitive in a graphics procedure, type and execute the following procedure. Note that you must press ENTER after completing each line, even if the cursor has already dropped to the next line because it reached the end of the screen.

```
TO MOVE
      CLEAN
      SETSPLIT 2
      PRINT [HOW FAR DO YOU WANT THE]
      PRINT1 [TURTLE TO MOVE? 1-9]
      FD RC
END
```

The PRINT1 primitive in Line 3 of this procedure causes the RC prompt to appear on the same line as the previous text. Every time you execute the program, you can



```
FOREGROUND COLOR...
```

type in a number in the range 1-9, and the Turtle moves that many steps on the graphics screen.

Making Requests

You can use REQUEST, or RQ, in a procedure to enter any number of characters from the keyboard. To tell LOGO when you have completed your entry, press ENTER. The following short procedure lets you enter a command or procedure from the keyboard and immediately see its execution:

```
TO MOVEMORE
      RUN RQ
      MOVEMORE
END
```

After you execute the procedure, try typing various commands such as:

```
REPEAT 3 [FD 50 RT 120] ENTER
```

Because the last line reexecutes the procedure, you can continue typing commands as long as you wish. To exit the procedure, press BREAK.

The following procedure uses REQUEST to let you change the graphics screen background and foreground colors. It then prints a graphics design to show off your new colors.

```
TO CHANGE
      PRINT1 [FOREGROUND COLOR...]
      SETPC FIRST RQ
      PRINT1 [BACKGROUND COLOR...]
      SETBG FIRST RQ
      CS
      REPEAT 72 [FD 40 BK 42 RT 5]
      CHANGE
END
```

Both the READCHARACTER and the REQUEST primitives have many applications. They are used again and again in this manual and are essential to many programs.

Cleaning Up the Keyboard

At times you need to be sure that no data remains in the keyboard input buffer before you enter new data. The CLEARINPUT procedure accomplishes this. For instance, if you write a procedure to delete unwanted files from your storage diskette, use the CLEARINPUT primitive to be sure that LOGO understands what you want and does not destroy a valuable file. The following procedure demonstrates this:

```
TO KILL
     PRINT1 [NAME OF FILE TO
      DELETE?..]
     MAKE "FILE FIRST RQ
     PRINT [I AM DELETING] :FILE
     PRINT1 [IS THIS OK?]
     CLEARINPUT
     IF RQ= [YES] [ERASEFILE :FILE
     PRINT :FILE [IS ERASED]]
          ELSE [PRINT :FILE [NOT
          ERASED]]
     KILL
END
```

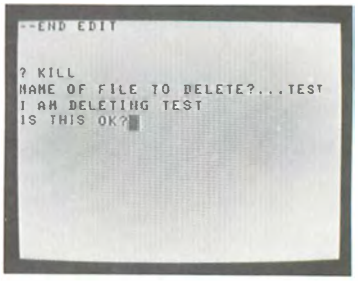
Using CLEARINPUT before requesting new keyboard input frees the keyboard buffer from any previous input that may cause problems.

Waiting for the Key

If your only concern is whether you pressed a key, and not what key you pressed, use the KEY? primitive. The KEY? primitive performs a true/false evaluation. The following procedure demonstrates how KEY? functions:

*. . . . **About the Keyboard Buffer***

Characters representing the keys you type are stored in a portion of your computer's memory called a keyboard buffer. *D.L. LOGO constantly scans this buffer for keyboard input. You observe the keyboard buffer in use if you type when LOGO is busy with a job that does not let it check the buffer. The keys you type do not appear on the screen until LOGO finishes the job and is free to read the buffer.*



```
--END EDIT

? KILL
NAME OF FILE TO DELETE?...TEST
I AM DELETING TEST
IS THIS OK?
```

```
TO KE
     IF KEY? [END]
     RT 45 FD 90 RT 184 FD 90 LT 180
     KE
END
```

This procedure continues to execute until you press a key. In effect, Line 1 of the procedure reads: *If a key is pressed, end the procedure.*

Chapter 3 Summary

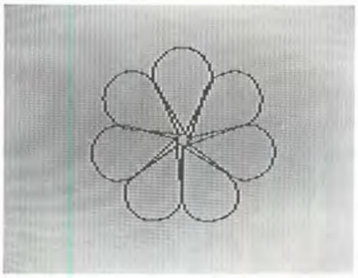
PRIMITIVE	Abbrev.	Purpose
TO	–	Specifies a procedure name.
END	–	Indicates the conclusion of a procedure.
EDIT	–	Causes LOGO to enter the edit mode.
PRINT	–	Prints specified text characters on the text screen, followed by a carriage return.
PRINT1	–	Prints specified text characters on the screen but does not produce a carriage return.
READCHARACTER	RC	Accepts 1-key input from the keyboard.
REQUEST	RQ	Accepts multiple-key input from the keyboard.
CLEARINPUT	–	Clears the keyboard buffer of all previous characters.
KEY?	–	Determines if a key is pressed.

Turtle Facts

- Executing procedures written in the edit mode does not destroy them. You can execute them as often as you wish.
- You must define procedure names with the primitive TO.
- Pressing ENTER ends a procedure line.
- Procedure names can be as long as you wish.
- D.L. LOGO ignores any text in a procedure that is preceded by a semicolon (;).

Editing Keys

Keys	Purpose
BREAK	Exits the insert and change modes. Pressing BREAK again exits the edit mode.
ENTER	Ends input of a procedure line. Positions the cursor at the beginning of a new line.
G	Moves the cursor to the beginning of the workspace.
SHIFT G	Moves the cursor to the end of text in the workspace.
I	Enters the insert mode. You remain in the insert mode until you press BREAK.
R *character*	Replaces the character under the cursor with the new character *character*.
X	Deletes the character under the cursor.
SHIFT R	Enters the change mode. You remain in the change mode until you press BREAK.
→	Tabs the cursor over 4 spaces to the right.
D D	Deletes one screen line of text from a procedure.
↑ ↓ → ←	Moves the cursor around the workspace.

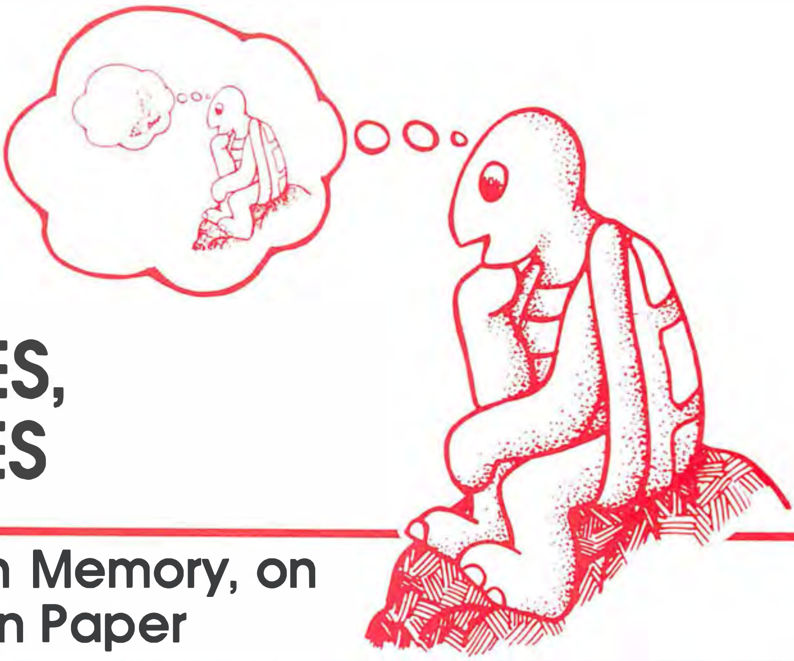


Suggested Project

Use the primitives and techniques you have learned to create a flower similar to the accompanying illustration. Remember, it is easier to break a program into small, simple procedures. In this case, you can break the assignment into 2 tasks: (1) to draw a petal and (2) to repeat the process enough times to create a flower. A suggested solution appears on the next page.

Suggested Project Solution

```
TO PETAL
     RT 69 FD 40
          REPEAT 30 [
          FD 3 RT 8]
     FD 40
END

TO FLOWER
     FULLSCREEN
     CS
     HT
     REPEAT 7 [PETAL]
END
```



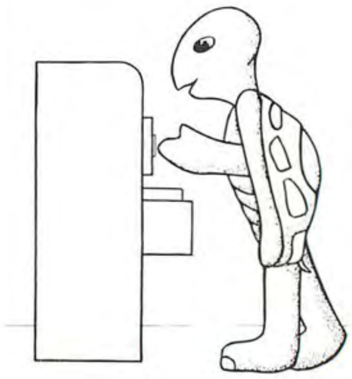
4 MEMORIES, MEMORIES

Procedures in Memory, on Diskette, or on Paper

Section 1	Saving and Loading: storing your work on diskette.
Section 2	Making Room: gaining more diskette space.
Section 3	A Vanishing Act: erasing, restoring, and manipulating procedures in the workspace.
Section 4	Printing Procedures: listing procedures and procedure names on the screen and to a printer.

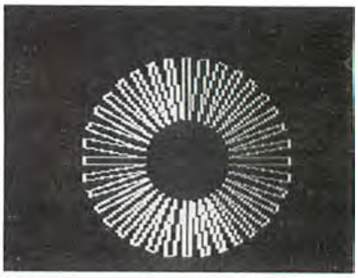
Section 1 Saving and Loading Procedures

You can save your D.L. LOGO work in 3 ways: storing it in the computer's memory, placing it on diskette, and making a paper copy with a printer. Procedures you type into the workspace are automatically saved in your computer's memory until you erase them, load other procedures from diskette, or turn off your computer. Procedures you save on diskette can be transferred to D.L. LOGO's workspace whenever you wish. Also, procedures stored on diskette are not erased when you turn off your computer.



Diskette Saving

To learn how to save your work, type the following 2 procedures. Execute them if you wish to see what they do.



```
    TO REC
         FD 40 RT 90
         FD 5 RT 90
         FD 40 RT 190
    END

    TO SPIN
         CS
         HT
         SETSPLIT 1
         PU FD 20 PD
         SETPC 3
         REPEAT 3[
         REPEAT 36 [REC]
         SETPC PC - 1]
    END
```

. . . . About Saving Procedures

*Whenever you save any procedure from LOGO's workspace to diskette, you automatically save all procedures in LOGO's workspace. For instance, if you have created 3 procedures, named PROONE, PROTWO, and PROTHREE, and you type **SAVE "PROTWO** ENTER, PROONE and PROTHREE are saved with PROTWO under the filename PROTWO.*

If you have only 1 disk drive and wish to save your programs and procedures on a diskette in Drive D0 (Drive 0), type:

```
     ? SAVE "name" ENTER
```

name represents the actual name you give the procedure or program you are saving.

If you have a second drive, save the program or procedure on a diskette in Drive D1 (Drive 1) by typing **SAVE "/D1/*name***. For this example, name your program SPIN and save it on Drive D0 by typing **SAVE "SPIN** ENTER.

To save the program on Drive D1, type **SAVE "/D1/SPIN** ENTER. To restore the SPIN program from the Drive D0 diskette to D.L. LOGO's workspace, type **LOAD "SPIN** ENTER. You can now edit, execute, or view the SPIN procedure in the normal manner.

D.L. LOGO follows the OS-9 convention of allowing the storage space on a diskette to be divided into any number of directories. Unless you indicate otherwise, D.L. LOGO saves files in the root directory. You can use OS-9 commands to create other directories on a diskette if you wish, but doing so is not necessary to fully utilize D.L. LOGO's capabilities.

If you do create other directories, you can use the CHD primitive to tell D.L. LOGO which directory is to be the current one. For example:

```
     CHD "TURTLEDIR
```

Changes the current directory to a directory named TURTLEDIR. If TURTLEDIR does not exist, an error message is displayed. See the OS-9 Commands manual for information on creating directories.

Reading the Catalog

To ensure you saved the program correctly, type the primitive **CATALOG** ENTER. The screen displays all of the programs or procedures saved on the current directory of the diskette in Drive D0. To see the contents of the current directory of a diskette in Drive D1, type **CATALOG "/D1** ENTER. If you have other directories on your diskette, you can view the contents of these by using quotation marks with the CATALOG primitive, such as (CATALOG "/D0/GRAPHICS).

If you have a large number of files on your diskette, you find that CATALOG causes some to scroll off the screen before you can read them. Because CATALOG can be an input to a variable (you learn about variables in Chapter 7), you can create a program to display diskette files. The CAT program in Appendix B of the manual is such a program. It gives you the options of displaying D.L. LOGO's file directory on the screen, or on the screen and to a printer. It prompts you for the drive number of the diskette you wish to display, then displays the files a screen full at a time. Press SPACE BAR to display subsequent files. If you select the printer option, turn the printer on before pressing **Y** at the prompt.

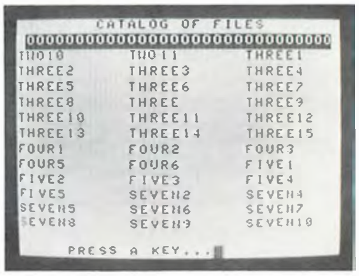
Many of the primitives and commands in this program are unfamiliar at this time. However, you may wish to add the program to your D.L. LOGO library for your convenience in examining the contents of D.L. LOGO's disk directory.

When you have typed CAT, save it on diskette by typing:

```
? SAVE "CAT ENTER
```

To use the program, type:

```
? LOAD "CAT ENTER
? CAT ENTER
```



```
            CATALOG OF FILES
000000000000000000000000000000000
TWO10          TWO11          THREE1
THREE2         THREE3         THREE4
THREE5         THREE6         THREE7
THREE8         THREE          THREE9
THREE10        THREE11        THREE12
THREE13        THREE14        THREE15
FOUR1          FOUR2          FOUR3
FOUR5          FOUR6          FIVE1
FIVE2          FIVE3          FIVE4
FIVE5          SEVEN2         SEVEN4
SEVEN5         SEVEN6         SEVEN7
SEVEN8         SEVEN9         SEVEN10

        PRESS A KEY...
```

. . . About Picture Files

Be careful that you do not save picture files using the same name as you used for program files. If you do, the picture files overwrite *the program files, destroying them.*

Then answer the prompts for the disk drive number and whether you wish a printer copy, as they appear on the screen.

Photo on a Disk

Not only can D.L. LOGO save procedures and programs, it can also save the graphics you create. After you save these *pictures*, you can restore them on the graphics screen or reproduce them on a printer with graphics capabilities. Although D.L. LOGO does not reproduce graphics screens on a printer, you can use Tandy's OS-9 High Res Screen Dump for this purpose. The screen dump manual tells you how to produce a printer copy of graphics screen files.

You can save a graphics screen from either the immediate or the graphics mode. To do so, type:

```
? SAVEPICT "name [ENTER]
```

name represents the actual name you gave the graphics picture. You must use the same name to reload the file. To save the design created by SPIN and REC, type **SAVEPICT "WHEELPIC** [ENTER]. To load the file back into LOGO's workspace, type **LOADPICT "WHEELPIC** [ENTER]. The design reappears on the graphics screen. Then you can use any LOGO graphics procedures to modify the graphics design and resave it to diskette.

If you have a second drive and wish to save the graphics screen on a diskette in Drive D1, type:

```
? SAVEPICT "/D1/WHEELPICT [ENTER]
```

To restore the file on the graphics screen, type:

```
? LOADPICT "/D1/WHEELPICT [ENTER]
```

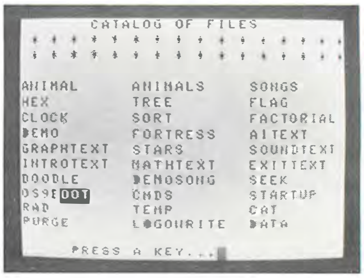
Section 2
Making Room

Checking Diskette Space

As you save the demonstration procedures and programs in this manual, as well as some of your own, you may be getting short of space on your diskette. Using the CATALOG primitive, you can check to see if there are any files that you no longer need. Then you can delete such files from your diskette, using D.L. LOGO's ERASEFILE primitive.

For instance, you may wish to erase some of the D.L. LOGO demonstration files. Using the CAT program from Appendix B, you can see the names of the files on the screen:

```
ANIMAL        ANIMALS      SONGS
HEX           TREE         FLAG
CLOCK         SORT         FACTORIAL
DEMO          FORTRESS     AITEXT
STARS         SOUNDTEXT    GRAPHTEXT
INTROTEXT     MATHTEXT     EDITTEXT
DEMOSONG      DOODLE
```



```
          CATALOG OF FILES

ANIMAL      ANIMALS     SONGS
HEX         TREE        FLAG
CLOCK       SORT        FACTORIAL
DEMO        FORTRESS    AITEXT
GRAPHTEXT   STARS       SOUNDTEXT
INTROTEXT   MATHTEXT    EXITTEXT
DOODLE      DEMOSONG    SEEK
OS9BOOT     CMDS        STARTUP
RAD         TEMP        CAT
PURGE       LOGOWRITE   DATA

      PRESS A KEY...
```

Caution: Do not erase files from your D.L. LOGO master diskette. It is always best to keep several backups of this diskette. Use the backup diskettes when running programs, copying, or deleting files.

To delete the first demonstration file, type **ERASEFILE "ANIMAL** ENTER.

Drive 0 starts, and after a short time, the LOGO cursor reappears. This tells you that the file has been erased. Follow the same procedure for as many files as you wish, but be careful. Once erased, a file cannot be retrieved. To erase files on other drives, specify the drive number. For instance:

```
    ? ERASEFILE "/D1/SONGS ENTER
```

If you plan to remove more than 1 or 2 files, a *purge* program is provided in Appendix B to make such a task easier.

Another Method of Erasing Files

Another way to remove an unwanted file is to save a new file under the existing name. Anytime you save a file, using a file name that already exists, the new file replaces the old file. Be careful not to use the name of an existing file you wish to keep when you save programs or procedures on your diskette.

Section 3
A Vanishing Act

As well as erasing and manipulating files on diskettes, you can also erase and manipulate procedures in the LOGO workspace. The ERASE and ERALL primitives remove a file or clear the entire workspace. If you have 4 procedures in the workspace named PICTURE, FRAME, STEM and FLOWER, and you decide you no longer need the FRAME procedure, type:

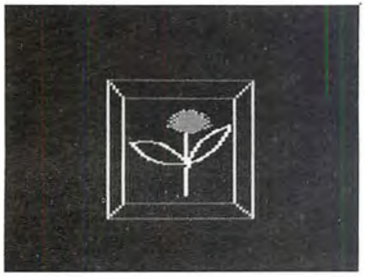
```
? ERASE "FRAME [ENTER]
```

The ERASE primitive removes FRAME from the workspace, leaving FLOWER and STEM.

Should you decide that you need FRAME after all, you can enter the edit mode by typing **EDIT** [ENTER], and press [U] to *undo* the last edit command. Pressing [U] reverses the previous ERASE primitive and restores FRAME. This *undo* function only operates on the last command you use in the edit mode. If you press any other keys before pressing [U], the *undo* function does not work on the erased files.

Test this process by entering the following procedures:

```
TO FRAME
     SETBG 12
     SETPC 2
     PU BK 50 LT 90 FD 50
     RT 90 PD
     REPEAT 4 [FD 95 RT 90]
     RT 45 FD 16 LT 45
     REPEAT 4 [FD 72 LT 45 FD 16
      BK 16 RT 135]
     PU FD 6 RT 90 FD 40 LT 90 PD
     STEM
END
```



```
    TO STEM
          SETPC 3
          FD 10
          REPEAT 10 [FD 5 RT 8]
          LT 68
          REPEAT 16 [RT 8 BK 5]
          RT 122
          REPEAT 8 [BK 5 RT 8]
          RT 38 FD 20
          FLOWER
    END

    TO FLOWER
          SETPC 1
          LT 84
          REPEAT 15 [RT 10 FD 15
           BK 15]
    END
```

Save the program to diskette then execute it by typing **FRAME** [ ]. When the program is complete, remove the FRAME procedure by typing:

```
? ERASE "FRAME [ENTER]
```

Now try to execute the FRAME program. Type **FRAME** [ENTER]. D.L. LOGO displays an error message telling you that FRAME is an undefined procedure. You can run the STEM and FLOWER procedures by typing **CS STEM** [ENTER].

To restore the FRAME procedure to the workspace, enter the edit mode by typing **EDIT** [ENTER], then press [U]. The cursor moves to the top of the workspace, and the FRAME procedure reappears.

The Total Erase

The ERALL primitive removes all files from the workspace. Typing EDIT [ENTER] after using the ERALL primitive displays a blank screen. However, you can restore the erased primitives by pressing U when in the edit mode. This is only possible if you have typed nothing else in the edit mode. After you type any other entry, it becomes the last edit input that U can affect and the erased files are lost.

Adding On

D.L. LOGO's APPEND primitive lets you load procedures into your workspace to the capacity of your computer's memory. Using this feature, you can maintain a library of procedures for use in any program, or write programs in modules that can be combined. To see how this works, load the previously saved SPIN program into the workspace by typing **LOAD "SPIN** ENTER.

When the ? prompt appears, type **ERASE "SPIN** ENTER.

Using the ERASE procedure eliminates the SPIN procedure from LOGO's workspace. Because the SPIN program consists of 2 procedures, REC and SPIN, you can save the remaining procedure under its name REC.

Again load SPIN. This time, delete the REC procedure by typing **ERASE "REC** ENTER. Save the remaining SPIN procedure by typing **SAVE "SPIN** ENTER. Doing this causes the new SPIN procedure to overwrite the previous SPIN file.

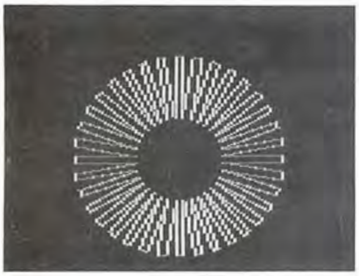
From the immediate mode, type **ERALL** ENTER. This erases all text from the workspace. Type **EDIT** ENTER to reenter the edit mode, and then type the following procedure:

```
TO SPIN2
    SETPC 1
    REPEAT 4[
        HOME
        SPIN
        SETPC PC+1]
END
```

*. . . . **About a Library***

A comprehensive disk library of procedures can save a programmer a great deal of time. If you write a program which contains one or more procedures you think you can use in future tasks, follow these steps to add it to a library diskette.

- *If you do not already have a library diskette, format a diskette to use.*
- *Save your program to your working diskette.*
- *Insert the library diskette.*
- *Erase all procedures except the one you wish to save from the workspace.*
- *Save the procedure.*



To save the procedure, type:

```
? SAVE "SPIN2 [ENTER]
```

You now have 3 separate procedures that can act together to create a program. To join the parts, type the following commands:

```
? ERALL [ENTER]
? LOAD "SPIN [ENTER]
? APPEND "REC [ENTER]
? APPEND "SPIN2 [ENTER]
```

Enter the edit mode to ensure that all the procedures are in the workspace. Exit the edit mode and execute the program by typing **SPIN2** [ENTER]. The 3 appended procedures work together to produce a new design.

Section 4
Printing Procedures

It is not necessary to enter the edit mode to examine procedures. The PRINTOUT, POALL, and POTS primitives display procedures and names of procedures on the immediate mode screen. To examine a particular procedure, use the abbreviated form of PRINTOUT. If you have the procedure REC in the workspace, type **PO "REC** and the screen shows:

```
    TO REC
         FD 40 RT 90
         FD 5 RT 90
         FD 40 RT 190
    END
```

Printing Names

To see the names of all the procedures currently in the workspace, use the POTS primitive. If you have several procedures in the workspace, the POTS primitive provides a printout resembling the following:

```
    ? POTS
    REC
    SPIN1
    SPIN2
    BEEP
    NOISE
    QUIT
    LOW
```

The Whole Show

The POALL primitive lets you view the entire contents of the workspace. This primitive sends procedure names and listings to the display screen. If you have more text in the workspace than 1 screen can display, text scrolls off the screen until the end of the workspace is reached.

Using A Printer

If you connect a printer to your computer, you can use the printout functions already mentioned in conjunction with the COPYON primitive. COPYON sends all screen displays to the printer. In this manner, you can make hard copies of procedure listings, names of procedures, or the entire workspace. The following examples show how to do this:

COMMANDS	RESULT
? COPYON ENTER ? POALL ENTER ? COPYOFF ENTER	–prints the entire workspace
? COPYON PO "DRAW COPYOFF ENTER	–prints the DRAW procedure
? COPYON POTS ENTER ? COPYOFF ENTER	–prints the names of all procedures resident in the workspace

You can combine the various printer and printout commands on 1 line or you can use separate lines as demonstrated in the preceding chart.

Printout from Procedures

You can also send data produced by procedures to a printer. If a command generates an output, such as PRINT :A + :B, you can use COPYON to send the data to a printer as well as to the display screen. The following example shows this:

```
TO SEND
     COPYON
     DO [
     MAKE "S RQ
     ]
     WHILE :S <> [END]
     COPYOFF
END
```

Each line you type goes to both the screen and the printer until you type "END". At this time, the COPYOFF primitive is activated and the procedure ends.

Chapter Summary

Primitive	Abbrev.	Purpose
SAVE	–	Copies a program or procedure to diskette.
LOAD	–	Copies a program or procedure from diskette to LOGO's workspace.
CATALOG	–	Displays all diskette files.
CHD	–	Changes the current directory.
SAVEPICT	–	Copies the contents of a graphics screen to diskette as a picture file.
LOADPICT	–	Displays a picture file from diskette on the graphics screen.
ERASEFILE	–	Removes a specified file from a diskette.
ERASE	–	Removes a specified procedure from the workspace.
ERALL	–	Removes all current procedures from the workspace.

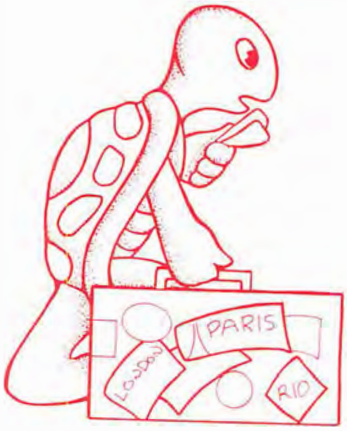
Primitive	Abbrev.	Purpose
APPEND	–	Loads the specified diskette file below the procedure or procedures currently in the workspace.
PRINTOUT	PO	Displays all lines of the specified procedure.
POALL	–	Displays the lines of all procedures currently in the workspace.
POTS	–	Displays the names of all procedures currently in the workspace.
COPYON	–	Causes a dual output of all screen display to the printer.
COPYOFF	–	Turns off COPYON.

Turtle Facts

- You can save D.L. LOGO work with 3 devices: the computer's memory, diskettes, and a printer.
- You can save files on D.L. LOGO diskettes, or on any OS-9 formatted diskettes.
- You can save graphics displays on a diskette and use a special screen dump utility to reproduce them on a printer equipped with graphics capabilities.
- If you save a new file using the name of a file currently on diskette, the old file is destroyed.

- You can save individual procedures in a program with the SAVE, LOAD, and ERASE primitives.
- You can make hard copy listings of procedures using the COPYON primitive.

5 COMPLETING THE TOUR



More Turtle Graphics

Section 1	More Geography: advanced primitives for directing the Turtle.
Section 2	More Turtle Maneuvers: directing the Turtle, controlling the pen, using dots, and hiding the Turtle.
Section 3	Building Fences and Roaming Free: restricting the Turtle to the borders of the screen, wrapping it around the edges, or sending it beyond the visible boundaries.

Section 1
More Geography

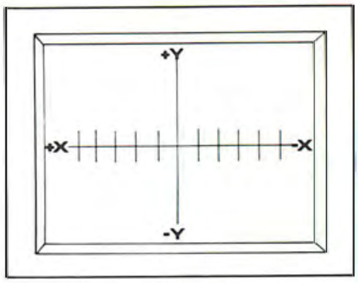
Now that you are familiar with the process of writing, saving, and printing procedures, you can create graphics routines for your Turtle with greater confidence. Once a procedure is working well, save it on diskette, and then try modifications and experiments. If you have a printer, make *hardcopies* of procedures so that you can more easily spot routines that need correcting or changing. Knowing that you will not destroy or erase the original procedure gives you more confidence to explore and test LOGO's capabilities. This chapter completes the introduction of the D.L. LOGO graphics capabilities begun in Chapter 1. Now, however, you need not lose the procedures and programs you create.

Teleporting Turtle

D.L. LOGO has a number of primitives to help your Turtle find its way around the graphics screen. As mentioned in Chapter 1, your Turtle is familiar with the Cartesian coordinate grid system. Using this system, you can instantly teleport the Turtle to any point on the grid.

If you have a procedure requiring the Turtle to start at the top left corner of the graphics screen, you can easily plot the necessary coordinates as –128 96. To send the Turtle to that point, first enter the graphics mode by pressing SHIFT BREAK. Then, set the new Turtle location by typing **SETXY –128 96** ENTER. The Turtle immediately jumps to the top left corner of the screen.

You need not set both the X and Y coordinates at the same time. To move the Turtle to the top middle of the screen, set only the X coordinate by typing **SETX 0** ENTER.



. . . . About the Grid

The above illustration shows the layout of the Cartesian Grid System. The grid is divided into 4 quadrants by a horizontal X line and a vertical Y line. The X and Y lines are divided in two, at point 0. On the X line, values to the left of zero are negative and values to the right of zero are positive. On the Y line, values above zero are positive and values below zero are negative. To find any coordinate, draw an imaginary perpendicular line from the desired locations on the X and Y lines. The coordinate location is where the 2 lines cross.

. . . . About Screen Locations

A point on the graphics screen is defined as an ordered pair of numbers. The first number indicates the horizontal position (X) and the second number indicates the vertical position (Y). Both numbers are required to identify the point.

Now reset the Y coordinate back at the HOME position by typing **SETY 0** ENTER.

Calculating Coordinates

You can establish the location of the Turtle using the XCOR and YCOR primitives. For instance, if a procedure moves the Turtle around the screen, and you want to know when it reaches a certain point, you can use XCOR and YCOR to provide that information. Type and execute the following procedure to see how this works:

```
TO TATTLE
     SETSPLIT 5
     SETXY -128 96
     REPEAT 10 [
          SETXY XCOR+10 YCOR-10
          PRINT [TURTLE IS AT] XCOR
           YCOR]
     END
```

Press BREAK twice to exit the edit mode. Execute the procedure by typing **TATTLE** ENTER. The Turtle takes stairlike steps down the screen, telling you its location at each step. XCOR always gives the Turtle's X coordinate, and YCOR always gives its Y coordinate.

Turtle as an Author

Although you have been giving commands to the Turtle, so far it has not communicated with you. Your Turtle can communicate in several ways, one of which is in writing. Although the Turtle isn't a creative writer (you have to tell it what to write), it can display messages on the graphics screen. Your Turtle can:

- Label elements of a graph
- Write messages for your family and friends

. . . . About Teleporting

When you send the Turtle to the coordinates at −128 96, the Turtle image disappears from the screen. This is because the Turtle is at the exact corner of the display, the image is off screen. To cause the Turtle image to show, type:

```
RT 135 ENTER
```

The Turtle now points diagonally towards the bottom right corner of the screen and most of the image is visible.

. . . . About the Turtle Position

When you send the Turtle to a particular grid coordinate, it appears centered 1 step from that coordinate, in the direction of its heading. For instance, if the Turtle is facing up, LOGO centers it immediately above the current grid coordinate. If it is facing down, LOGO centers it immediately below the current grid coordinate, and so on. To test this, send the Turtle to some coordinate, and then display a dot at that coordinate. For instance, type:

```
? SETXY 10 10 DOT 10
 10
```

- Tell you what it is doing while it executes a procedure
- Give instructions during a game
- Give prompts and verify answers
- Label demonstrations

To create text on the graphics screen, use the TURTLETEXT primitive. To see how this works, execute the previous TATTLE procedure with one change:

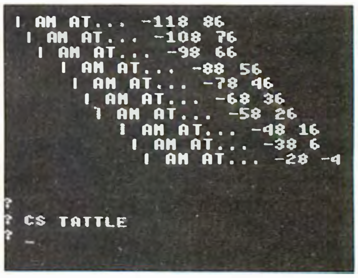
```
    TO TATTLE
          SETSPLIT 5
          SETXY -128 96
          REPEAT 10 [
                SETXY XCOR+10 YCOR-10
                TURTLETEXT [I AM AT...]
                 XCOR YCOR]
          END
```

The TURTLETEXT primitive displays a message at the current Turtle position on the screen.

. . . . About TURTLETEXT

When you use TURTLETEXT to display a message on the graphics screen, the message is placed with the upper left corner of the first letter on the specified grid coordinate. To test this, execute this command:

```
  SETXY 50 50 DOT 50 50
   TURTLETEXT [HERE I
   AM]
```



```
I AM AT... -118 86
 I AM AT... -108 76
  I AM AT... -98 66
    I AM AT... -88 56
     I AM AT... -78 46
      I AM AT... -68 36
       I AM AT... -58 26
         I AM AT... -48 16
          I AM AT... -38 6
           I AM AT... -28 -4


? 
? CS TATTLE
? _
```

. . . . About Powers

In D. L. LOGO, the up arrow (↑) indicates powers. For instance 4↑2 is interpreted as 4 to the power of 2. You learn about D. L. LOGO math functions in Chapter 9.

Section 2 More Turtle Maneuvers

Heading Out

D.L. LOGO has 2 commands that let your Turtle reach any point by the most direct route. For instance, if your Turtle is at home but you wish it to draw a line straight to coordinates 5Ø 5Ø, use the TOWARDS and SETHEADING primitives. Try the following procedure:

```
TO TRI
     CS
     FD 5Ø
     RT 9Ø
     FD 5Ø
     SETHEADING TOWARDS Ø Ø
     FD SQRT (5Ø↑2 + 5Ø↑2)
END
```

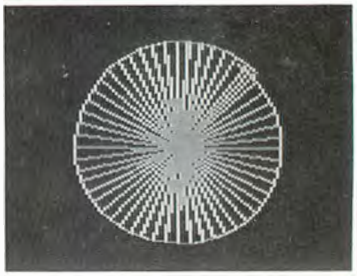
This procedure first draws 2 sides of a triangle and then uses the TOWARDS primitive to provide the direction for the SETHEADING primitive. Because the Turtle begins at coordinates Ø Ø, it ends there to complete the job.

The last line in the procedure uses a version of the formula C↑2 = A↑2 + B↑2 to calculate the distance from the second side of the triangle to home.

Try this procedure to see the power of the TOWARDS and SETHEADING primitives.

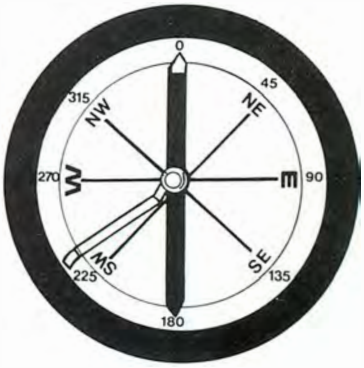
```
TO RAD
     CS
     FULLSCREEN
     SETXY 5Ø 5Ø
     REPEAT 54 [
                 SETHEADING TOWARDS 0 0
                 FD 71 BK 71
                 RT 86
                 FD 8.2]
     END
```

There are easier and quicker ways to draw radiating lines in a circle, but this procedure illustrates the Turtle's ability to always find its way to an exact coordinate, regardless of its current position on the screen.



The HEADING primitive lets you check the Turtle's current heading at any time. When the previous procedure ends, type **HEADING** ENTER. The screen shows a value of 321. This gives you another method of creating radiating lines. Try this procedure:

```
     TO SUN
          FULLSCREEN
          CS
          REPEAT 45 [ FD 71 BK 71
          SETHEADING HEADING + 8 ]
     END
```



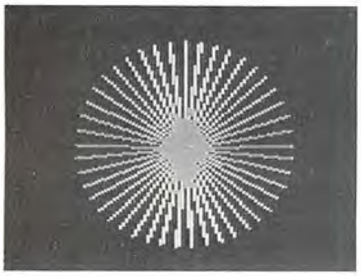
This time the Turtle uses coordinates 0 0 as home base and uses SETHEADING to increment the value HEADING returns. Heading sets are similar to degrees on a compass; directly up is heading 0. Heading values increment clockwise to 360 degrees, as illustrated:

. . . . ***About Headings***

Turtle headings directly relate to compass headings. For instance, to cause the Turtle to face right, set the heading to 90. To cause the Turtle to face left, set the heading to 270, and so on.

To Pen or NOT to Pen

Although your Turtle's pen never runs out of ink, at times you may want to move the Turtle without leaving a trail. You can do this with the PENUP primitive. Later, when you want the Turtle to draw again, use the PENDOWN primitive. To test these primitives, change the previous RAD procedure by adding these lines:



```
TO RAD
      CS
      FULLSCREEN
      SETXY 50 50
      REPEAT 54
      [SETHEADING TOWARDS 0 0
      FD 71 BK 71
      RT 86
      PENUP
      FD 8.2
      PENDOWN]
END
```

By lifting the pen when the Turtle goes around the circle and putting it down for the radiating lines, you can create a design that looks like the SUN design.

You can also check the condition of the pen. To see if the pen is down, use the PENDOWN? primitive. If the pen is down, the screen displays TRUE; if the pen is up, the screen displays FALSE. For example:

```
? PENDOWN? ENTER
TRUE
```

DOT Marks the Spot

LOGO also lets you mark locations on the screen without your Turtle leaving home. The DOT primitive produces a dot at a specified location on the grid. To see this, type:

```
? HOME ENTER
? DOT 10 10 ENTER
? DOT -10 -10 ENTER
? DOT -10 10 ENTER
? DOT 10 -10 ENTER
```

If you now type **PRINT XCOR YCOR**, you see that, despite the new dots on the screen, Turtle is resting at home.

A Disappearing Act

Sometimes your Turtle gets in the way of a graphics design or graphics text. You can tell it to disappear with the HIDETURTLE or HT primitive. Type and execute the following procedure:

```
TO SPOT
      HIDETURTLE
      PENUP
      REPEAT 5[
            REPEAT 20 [DOT XCOR YCOR
             FD 2]
            RT 144]
      SHOWTURTLE
END
```

You have probably guessed that SHOWTURTLE is the primitive that gets the Turtle out of hiding. Both primitives appear in their long forms in this procedure, but HT and ST work too. In this procedure, the FD primitive moves the Turtle, and the XCOR and YCOR primitives tell the DOT primitive where to execute its function.

Efficiency is another reason to use the HIDETURTLE command when you execute graphics procedures. To display the Turtle during the execution of a procedure, D.L. LOGO constantly redraws the Turtle shape on the screen. If you hide the Turtle, LOGO has more time to operate your procedure and functions much faster. To see the extra speed HIDETURTLE gives, execute the following procedure twice, once with the Turtle showing and once with it hidden:

```
TO BAR
      CS PD
      REPEAT 30
            [FD 80 RT 90 FD 1 RT 90
            FD 80 LT 90 FD 1 LT 90]
END
```

To execute the procedure with the Turtle, type **CS ST BAR** ENTER. Time the procedure if you like. Now type **CS HT BAR** ENTER. The second execution runs approximately one-third faster. Restore the Turtle on the graphics screen by typing **ST** ENTER.

If you are in the immediate mode and want to know whether the Turtle is hidden or not, you can use the SHOWN? primitive to tell you the Turtle's state. Type **SHOWN?** ENTER. If the Turtle is visible, TRUE appears; if the Turtle is hidden, FALSE appears on the screen. You can also use SHOWN? from within procedures to test the Turtle state.

Section 3 Building Fences and Roaming Free

D.L. LOGO can fence your Turtle within the borders of the graphics screen, let it wrap around the edges, or give it freedom to wander beyond visible bounds. The 3 primitives that do this are FENCE, WRAP, and WINDOW, respectively.

When you first load D.L. LOGO, WINDOW is active. If you issue a command to send the Turtle beyond the border of the screen, you lose sight of it. You can do this when the Turtle is in the HOME position by issuing the command:

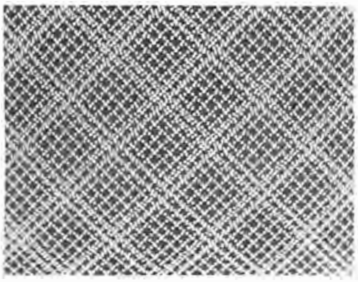
```
? FD 200 ENTER
```

The Turtle disappears from the top of the screen. As long as it is on the screen, you can see the trail left by the Turtle. Once off the screen, however, the trail continues into oblivion, and the Turtle seems lost. To return the Turtle to the screen, use a command such as HOME or CLEARSCREEN; establish new XY coordinates; or issue a WRAP primitive.

WRAP, in effect, connects 1 edge of your screen to the opposite edge. Thus, when your Turtle goes off the edge at the top, it immediately reappears at the bottom. Or, if it goes off the edge at the right side, it reappears at the opposite edge of the left side. To see this, issue the FD 200 command again, but before you do, use the WRAP primitive:

```
? CS WRAP FD 200 ENTER
```

This time the Turtle trail goes to the top of the screen and reappears at the bottom. The Turtle completes its mission slightly above the center of the screen.



For a striking illustration of the WRAP primitive, execute this procedure:

```
TO WEAVE
      FULLSCREEN
      CS
      WRAP
      REPEAT 10
            [SETPC 1
            RT 45
            FD 1090
            SETPC 2
            RT 90
            FD 1090
            SETHEADING 90
            PU FD 4 PD
            LT 90]
END
```

By turning the Turtle right 45 degrees and sending it forward 1090 steps, you create an effect similar to winding tape down a stick. Of course, this works only if the WRAP primitive is active. If you wonder what the procedure does without the WRAP primitive, replace it with WINDOW and reexecute the procedure.

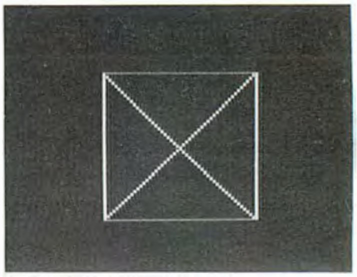
To prevent the Turtle from leaving the screen boundaries, use FENCE. When you implement the FENCE command and the Turtle reaches an edge of the screen, the current procedure stops and an error message says, ** ERROR: TURTLE OUT OF BOUNDS.

Chapter Summary

PRIMITIVE	Abbrev.	Purpose
SETXY		Establishes graphics screen X and Y coordinates.
SETX	–	Establishes the graphics screen X coordinate.
SETY	–	Establishes the graphics screen Y coordinate.
XCOR	–	Returns the current graphics screen X coordinate.
YCOR	–	Returns the current graphics screen Y coordinate.
TURTLETEXT	–	Displays a message on the graphics screen at the current x/y grid coordinates.
TOWARDS	–	Provides the degree heading from Turtle's current position to a specified position.
SETHEADING	–	Sets the heading of the Turtle in the range of 0-360 degrees.
HEADING	–	Gives the Turtle's current heading.

PRIMITIVE	Abbrev.	Purpose
PENUP	PU	Disables the Turtle pen.
PENDOWN	PD	Enables the Turtle pen.
PENDOWN?	–	Returns the condition of the pen. Down = TRUE, Up = FALSE.
DOT	–	Displays a dot on the graphics screen at specified coordinates.
HIDETURTLE	HT	Causes the image of the Turtle to disappear from the graphics screen.
SHOWTURTLE	ST	Causes the image of the Turtle to reappear on the graphics screen.
SHOWN?	–	Returns TRUE if the Turtle is visible, FALSE if it is hiding.
FENCE	–	Causes a procedure to halt and display an error message if the Turtle attempts to go beyond the graphics screen boundaries.
WRAP	–	Causes graphics to reappear on the opposite side of the screen when they go beyond a screen boundary.

PRIMITIVE	Abbrev.	Purpose
WINDOW	–	Disables FENCE and WRAP. Displays only on-screen points.



Turtle Facts

- LOGO locates the Turtle headings in degrees, in the range of 0 to 360, with straight up being 0 degrees.
- Headings increment clockwise.
- The middle of the Turtle's back is always centered on its current grid coordinates, regardless of its heading.
- Graphics procedures run dramatically faster when the HIDETURTLE primitive is active.
- When you implement WINDOW, any Turtle graphics beyond the parameters of the graphics screen are not visible.

Suggested Project

Write a program to:

1. Arrange 4 dots in a square, each with an equal distance from the center of the screen.
2. Create a box by connecting the corners.
3. Use SETHEADING and TOWARDS to connect opposite corners and form an X in the box, similar to a ballot box.

Hint: the formula for calculating the length of the third side of a triangle is SIDE3↑2 = SIDE1↑2 + SIDE2↑2.

Suggested Project Solution

```
TO BALLOT
      CS
      FULLSCREEN
      CORNER
      BOX
      CROSS
END

TO CORNER
      DOT 50 50
      DOT -50 -50
      DOT 50 -50
      DOT -50 50
END

TO BOX
      SETXY -50 50
      REPEAT 4
            [RT 90 FD 100 ]
END

TO CROSS
      SETXY 50 50
      SETHEADING TOWARDS -50 -50
      FD SQRT (100↑2 + 100↑2)
      SETXY -50 50
      SETHEADING TOWARDS 50 -50
      FD SQRT (100↑2 + 100↑2)
END
```

6 YOU SING, TURTLE PLAYS

Turtle Music

Section 1	Technical Talk: introducing Turtle's musical talent.
Section 2	Getting Started: how Turtle makes music.
Section 3	Everybody Sing: using all 4 voices.

Section 1 Technical Talk

Your Turtle is a versatile musician. Whether you want "Home on the Range" or Wagner's "Ride of the Valkyrie," Turtle can oblige. But . . . you must enter the notes in a form that D.L. LOGO understands.

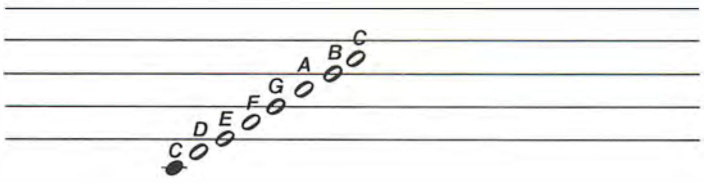
You don't need special equipment to produce D.L. LOGO music. Music generated by circuitry in your computer plays through your television speaker—and in 4-part harmony if you choose.

You can make music with D.L. LOGO even if you know very little about music. This chapter includes instructions that tell you how to convert musical notes to D.L. LOGO code.

Pitch, Octaves, and Rhythm

The musical scale consists of the notes C, D, E, F, G, A, and B, and each can be followed by a sharp (#) or a flat (♭).

Musical Score



In D.L. LOGO you use the pound symbol (#) to indicate a sharp, and you use the apostrophe (') to indicate a flat. To show that a voice is resting, use the letter X. Use the ampersand (&) to indicate that a specified voice is the same as its previous setting.

D.L. LOGO identifies octaves as beginning and ending on C. The octave beginning on middle C is the default octave. That is, unless you specify otherwise, D.L. LOGO uses the octave beginning at middle C when playing a note. You specify octaves higher or lower than the default octave by preceding a note designation with 1 or more H's or L's.

For example, LLLC tells D.L. LOGO that you want to play C that is 3 octaves below middle C. HC tells D.L. LOGO that you want to play C that is 1 octave above middle C.

Type the following line and listen to the sounds D.L. LOGO produces:

```
MUSIC [T360 LLLC LLC LC C HC HHC
  HHHC] ENTER
```

You must tell D.L. LOGO the number of beats to hold each note. You do so by preceding the name of the note with a number—1 tells D.L. LOGO to hold the note 1 beat, 2 to hold the note 2 beats, and so on.

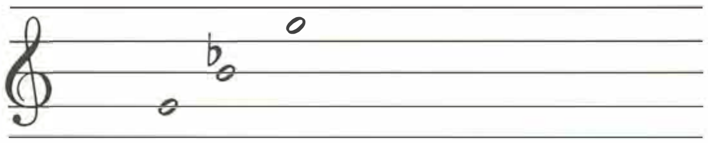
You must also specify the tempo, or the number of beats per minute. To do so, type T*n* (*n* is the number of beats). For example, to indicate 120 beats per minute, type **T120**.

In D.L. LOGO a *note-word* consists of from 1 to 4 notes that are sounded simultaneously. For example, a G minor chord (G B-flat D) consists of 3 notes sounded simultaneously and is therefore a note-word. In D.L. LOGO code the note-word is GB'HD, beginning on the G above middle C. Notice that D is preceded by an H, which tells D.L. LOGO that you want the D in the second octave above middle C. To hear this chord, type:

```
MUSIC [GB'HD] ENTER
```

Following are the same notes shown as notes on a staff.

Musical Score



Do-Re-Me

Now, to hear some real music, enter the following lines from the edit mode:

```
    TO DOREME
     MUSIC [T360 3C D 3E C 2E 2C 4E
                 3D
             E F F E D 8F 3E F 3G E
                 2G
             2E 4G 3F G A A G F 8A
                 3G
             C D E F G 8A 3A D E F#
                 G A
             8B 3B E F# G# A B 6HC
                 HC B
             2A 2F 2B 2G 5HC 4X C D
                 E F
             G A B HC X G X C]
    END
```

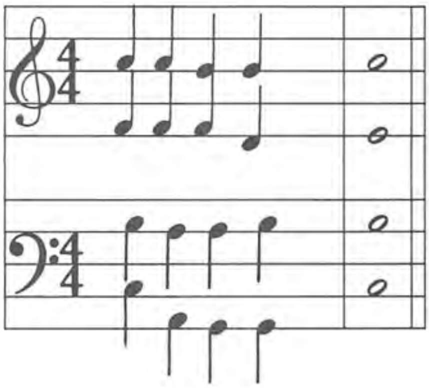
If all is well, you hear D.L. LOGO's rendition of "Do-Re-Me." If it doesn't sound quite right, check your typing with the listing and correct any errors.

Section 2 Getting Started

If you read music, the preceding information is probably all you need to to play jazz, pops, rock, blues, classical, or country and western in D.L. LOGO. If you don't read music, you can still tickle D.L. LOGO's ivories by using the guidelines that follow.

Following is a sample worksheet. On the left is a series of chords in two measures. On the right is the D.L. LOGO code (one "box" with a sum line for each measure in the score). Reading the top row of notes across, you have C C B B. Converting these notes to D.L. LOGO code produces 1HC 1HC 1B 1B. The second row of notes corresponds to the second row of code, as do the third and fourth rows. The bottom line of D.L. LOGO code is a *sum* line, or, in other words, the note-word. The note-word contains the 4 notes that sound simultaneously.

Musical Score



HC	HC	B	B
E	F	F	D
LG	LF	LF	LG
LC	LLA	LLA	LLG

HCELGLC HCFLFLLA BFLFLLA BDLGLLG

HC	HC	HC	HC
E	E	E	E
LG	LG	LG	LG
LC	LC	LC	LC

4HCELGLC

To hear this chord progression, you type in the sum lines as shown below each box of code. Because the last measure consists of whole notes, it can be entered as one noteword. You learn more about this later in the chapter. For now, type:

```
    MUSIC [T360 HCELGLC HCFLFLLA BFLFLLG
     BDLGLLG HCELGLC] ENTER
```

At the end of the chapter is a worksheet that you can use to help you convert musical scores to D.L. LOGO code. At the top left is a grand staff showing the default octave and the octaves immediately above and below. If you don't read music, you can use this chart to determine the names of the notes and their values. You can use the blanks below the grand staff to insert the corresponding D.L. LOGO code. Enter the codes in the appropriate squares, and then compile them into note-words. If you are writing your own music, use a pencil and eraser and experiment as you go.

Feel free to copy or reproduce this worksheet in any manner you like. Copyright laws forbid the copying of other portions of the manual.

A Bit of Music Background

If you are not familiar with music terminology, the following definitions will help you.

Term	Definition
Note	A symbol representing both a value and a duration. For example, a whole note represents a duration of 4 quarter notes. Its pitch is determined by its position on the staff.
Staff	The lines and spaces on which notes are placed. The grand staff consists of the treble clef staff and the bass clef staff.

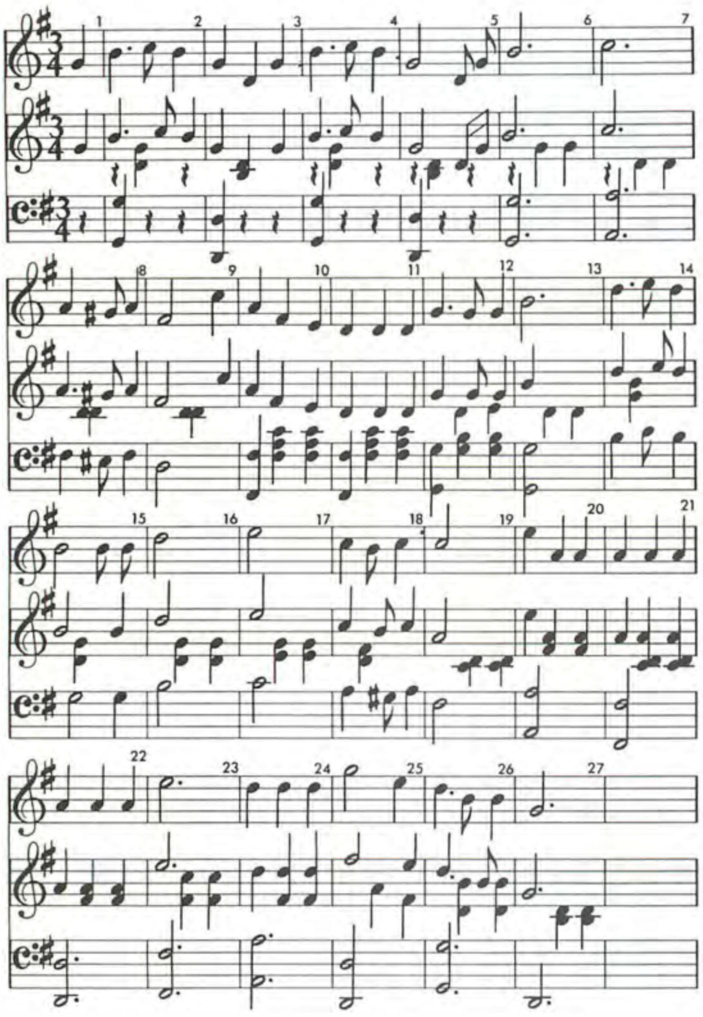
Term	Definition
Pitch	The tonal value of a note. The higher a note is placed on the staff, the higher its tonal value. The lower its placement, the lower its tonal value.
Treble clef	The upper staff. A treble clef staff is indicated by the treble clef symbol (𝄞).
Bass clef	The lower staff. The bass clef staff is represented by the bass clef symbol (𝄢).
Octave	A range of 8 notes.

Blow The Man Down

When you tell D.L. LOGO to make music, you must specify the value of each note (whether it is a whole note or a half note or a quarter note, and so on) and what letter represents its pitch. You must also designate any notes that are sharped or flatted.

You can use the following version of "Blow the Man Down" to practice making music with D.L. LOGO.

Blow the Man Down



This musical score has 2 treble clef staves and 1 bass clef staff. The top staff is the vocal line. For this rehearsal, convert only the vocal line to D.L. LOGO code. Later, you can enter the full score.

LOGO in Three-Quarter Time

The time signature in a piece of music is always indicated at the beginning of the piece by numbers that look like a fraction. The time signature for "Blow the Man Down" is 3/4.

This means that each measure (a measure consists of the notes between 2 vertical lines that intersect the staff) contains 3 beats and that a quarter note (♩) gets 1 beat. Look through the lines of music, and you'll see that each measure contains a total note value of 3 quarter notes. To interpret this piece of music for D.L. LOGO, break each measure into its smallest components. Look again through the piece. The smallest component is an eighth note (♪), which becomes the basic unit of time for this example. The worksheet for "Blow the Man Down" has 6 blanks for each measure—1 blank for each eighth-note value.

The Music

The first note in "Blow the Man Down" is a pickup quarter-note. (The measure does not have the full 3 quarter-note value.) To keep your chart in perspective, write this note in Line 1 below blanks 5 and 6. Although G is a quarter note, enter it as 2 eighth-notes to indicate each beat.

The first full measure consists of a dotted quarter-note B, an eighth-note C, and a quarter-note B. A dot after a note indicates that the note is extended 1/2 its original value; thus, the dotted quarter note has the same value as 3 eighth-notes. Enter the measure on Line 2 of the chart as 3 eighth notes on B, an eighth note on C, and 2 eighth notes on B. C is preceded by an H, because this C is 1 octave higher than the default octave.

The key signature (an arrangement of sharps or flats following the clef sign) shows that this piece has one sharp—F. This means that all the F's in "Blow the Man Down" are sharped. Be sure to include the D.L. LOGO sharp sign (#) for all F's that you convert to code.

Complete the code for the first measure by combining the notes according to their values. For example, the 2 pickup G's are played as 1 note, and D.L. LOGO needs to know this. Immediately above the 2 G notes, write 2G. The first 3 B's are also played as 1 note; write 3B above these notes. The fourth note (an eighth-note C) is separate; so write 1HC above this note. Write 2B above the last 2 notes. When entered into D.L. LOGO, the code looks like this:

```
2G
3B 1HC 2B
```

You can see that the note values of the first full measure add up to 6 eighth-notes, or the 3/4 value required for a measure. When only 1 beat is indicated, as in 1HC, you do not need to include the number 1. The following examples include the number 1 because later code changes require the inclusion of 1 before all 1-beat notes.

Because you are converting only the vocal line to code, the following worksheet shows only 1 line for each measure. Later examples include the 5 lines indicated on the original worksheet. Each line is numbered on the worksheet, and each measure is numbered on the score. This numbering is not usual on musical scores, but we have done it, and you can do it, to facilitate the conversion process.

1					G	G
2	B	B	B	HC	B	B

3	G	G	D	D	G	G
4	B	B	B	HC	B	B
5	G	G	G	G	D	G
6	B	B	B	B	B	B
7	HC	HC	HC	HC	HC	HC
8	A	A	A	G#	A	A
9	F#	F#	F#	F#	HC	HC
10	A	A	F#	F#	E	E
11	D	D	D	D	D	D
12	G	G	G	G	G	G
13	B	B	B	B	B	B
14	HD	HD	HD	HE	HD	HD
15	HD	HD	HD	HD	HD	HD
16	B	B	B	B	B	B
17	HE	HE	HE	HE	HE	HE
18	HC	HC	HC	B	HC	HC
19	A	A	A	A	A	A
20	HE	HE	A	A	A	A
21	A	A	A	A	A	A
22	A	A	A	A	A	A
23	HE	HE	HE	HE	HE	HE
24	HD	HD	HD	HD	HD	HD
25	HF#	HF#	HF#	HF#	HE	HE
26	HD	HD	HD	B	B	B
27	G	G	G	G	G	G

After you transfer the score to worksheets, combine the measures into note-words. The completed score, entered into D.L. LOGO, looks like this:

```
TO BLOW
     MUSIC [T360
     2G
     3B 1HC 2B
     2G 2D 2G
     3B 1HC 2B
     4G 1D 1G
     6B
     6HC
     3A 1G# 2A
     4F# 2HC
     2A 2F# 2E
     2D 2D 2D
     3G 1G 2G
     6B
     3HD 1HE 2HD
     4B 1B 1B
     6HD
     6HE
     3HC 1B 2HC
     6A
     2HE 2A 2A
     2A 2A 2A
     2A 2A 2A
     6HE
     2HD 2HD 2HD
     4HF# 2HE
     3HD 1B 2B
     6G]
END
```

To hear the tune, press BREAK twice to exit the edit mode and then type **BLOW** ENTER.

Refining the Score

You may have noticed that some notes blend into 1 long note. This is because D.L. LOGO plays adjacent notes as

. . . . *About Code Lines*

You need not put each measure's code on a separate line. You can arrange the code in any manner, including 1 long, continuous line. In this manual, we arrange the code with 1 measure per line to make checking and editing easier.

1 long note when there is not a value change between notes. You can correct this by inserting very short pauses between each note. An eighth-note pause is too long; the solution is to increase the number of beats per measure without changing the length of time each note is played. Because you can use any value for D.L. LOGO's tempo, you can multiply both the tempo value and the note values by 10 and set the tempo to 1200 (T1200). Now 10 beats are required to produce notes of the same duration as a setting of T120.

A pause in music is known as a rest, and it can be of the same duration as any note. Inserting the D.L. LOGO rest symbol (X) between note-words creates a rest. If you use X without a number prefix, the default value is 1. A duration of 1 with a tempo of 1200 produces a very slight, but distinct pause.

Search and Replace

You can use D.L. LOGO's global search and replace feature to insert pause symbols and change beat values. From the edit mode, begin the insertions by typing:

```
/G/G X/G ENTER
```

The first slash indicates a search. The following *G* is the character to find. *G* and *X* are the replacement characters, and the final */G* tells D.L. LOGO to do a global (all inclusive) search and replace.

Every G in the tune is now followed by *space X* (X). G# poses a special problem because the *X* is inserted in the wrong place. To put the *X* in proper relationship to this note, first delete the added *space X*. Now use global search and replace to add a *space X* after each sharp symbol. Type:

```
/#/# X/G ENTER
```

This also inserts the proper pause after all F sharps. Use the global search and replace method to insert pauses after A, B, C, D, and E. Insert an additional 0 after T360 (T3600).

Using global search and replace, add a 0 after each 1, 2, 3, 4, and 6. To do so, use the following command:

```
/value/value0/G ENTER
```

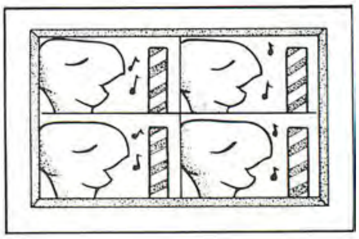
value is the number change. For example, to add a 0 to all 1's, type **/1/10/G** ENTER. Using the global search and replace function causes some problems. The procedure name (TO BLOW) and the MUSIC and END primitives have some extra X characters. As well, the tempo (T3600) now reads T306000. Delete these extra characters and you are ready to hear your revised version of "Blow the Man Down."

The revised score looks like this:

```
TO BLOW
     MUSIC [T3600
     20G X
     30B X 10HC X 20B X
     20G X 20D X 20G X
     30B X 10HC X 20B X
     40G X 10D X 10G X
     60B X
     60HC X
     30A X 10G# X 20A X
     40F# X 20HC X
     20A X 20F# X 20E X
     20D X 20D X 20D X
     30G X 10G X 20G X
     60B X
     30HD X 10HE X 20HD X
     40B X 10B X 10B X
     60HD X
     60HE X
     30HC X 10B X 20HC X
```

```
          60A X
          20HE X 20A X 20A X
          20A X 20A X 20A X
          20A X 20A X 20A X
          60HE X
          20HD X 20HD X 20HD X
          40HF# X 20HE X
          30HD X 10B X 20B X
          60G]
     END
```

Section 3
Everybody Sing



So far you have heard only the vocal line, or melody, of "Blow the Man Down." It's time to add the harmony.

The first measure, consisting of a pickup note, remains the same. You begin adding voices in the next measure. Your worksheet looks like this:

	B	B	B	HC	B	B
1.	LG	LG	G	G		
	LLG	LLG	D	D		

```
20BLGLLG            10BGD    10HCGD   20B
```

The B in the treble staff is a dotted quarter-note and requires 3 beats. Below this first note is a rest, which indicates that no note is played and need not be included on the chart.

In the bass staff are 2 G's. The first is 1 octave below the default octave (LG), and the second is 2 octaves below the default (LLG). These notes are quarter notes and require 2 eighth-note beats.

Two quarter-notes (G and D) begin on the second beat in the treble staff. Both require 2 beats; the first in conjunction with the third beat of the dotted quarter-note B, and the second in conjunction with HC. The last note is a quarter-note B and is played alone for 2 beats. The notes are combined into note-words of the proper value at the bottom of the chart.

Continue coding the rest of the measures. Be sure to include the sharp sign for all F's. To check the accuracy of your entries, compare them with the following chart.

G	G				

20G

1.

B	B	B	HC	B	B
LG	LG	G	G		
LLG	LLG	D	D		

20BLGLLG 10BGD 10HCGD 20B

2.

G	G	D	D	G	G
LD	LD	LB	LB		
LLG	LLD				

20GLDLLD 20DLB 20G

3.

B	B	B	HC	B	B
LG	LG	G	G		
LLG	LLG	D	D		

20BLGLLG 10BGD 10HCGD 20B

4.

G	G	G	G	D	G
LD	LD	D	D		
LLD	LLD	LB	LB		

20GLDLLD 20GDLB 10D 10G

5.

B	B	B	B	B	B
		G	G	G	G
LG	LG	LG	LG	LG	LG
LLG	LLG	LLG	LLG	LLG	LLG

20BLGLLG 20BGLGLLG 20BGLGLLG

6.

HC	HC	HC	HC	HC	HC
		D	D	D	D
LA	LA	LA	LA	LA	LA
LLA	LLA	LLA	LLA	LLA	LLA

20HCLALLA 20HCDLALLA 20HCDLALLA

7.

A	A	A	G#	A	A
		D	D		
		C	C		
LF#	LF#	LF#	LE#	LF#	LF#

20ALF# 10ADCLF# 10G#DCLE# 20ALF#

8.

F#	F#	F#	F#	HC	HC
		D	D		
		C	C		
LD	LD	LD	LD	LD	LD

20F#LD 20F#DCLD 20HCLD

9.

A	A	F#	F#	E	E
		C	C	C	C
LF#	LF#	LA	LA	LA	LA
LLF#	LLF#	LF#	LF#	LF#	LF#

20ALF#LLF# 20F#CLALF# 20ECLALF#

10.

D	D	D	D	D	D
		C	C	C	C
LF#	LF#	LA	LA	LA	LA
LLF#	LLF#	LF#	LF#	LF#	LF#

20DLF#LLF# 20DCLALF# 20DCLALF#

11.

G	G	G	G	G	G
		D	D	D	D
LG	LG	LB	LB	LB	LB
LLG	LLG	LG	LG	LG	LG

20GLGLLG 20GDLBLG 20GDLBLG

12.

B	B	B	B	B	B
		D	D	D	D
LG	LG	LG	LG	LG	LG
LLG	LLG	LLG	LLG	LLG	LLG

20BLGLLG 20BDLGLLG 20BDLGLLG

13.

HD	HD	HD	HE	HD	HD
		B	B		
		G	G		
LB	LB	LB	C	LB	LB

20HDLB 10HDBGLB 10HEBGC 20HDLB

14.

B	B	B	B	B	B
		G	G		
		D	D		
LG	LG	LG	LG	LG	LG

20BLG 20BGDLG 20BLG

15.

HD	HD	HD	HD	HD	HD
		G	G	G	G
		D	D	D	D
LB	LB	LB	LB	LB	LB

20HDLB 20HDGDLB 20HDGDLB

16.

HE	HE	HE	HE	HE	HE
		G	G	G	G
		E	E	E	E
C	C	C	C	C	C

20HEC 20HEGEC 20HEGEC

17.

HC	HC	HC	B	HC	HC
		F	F		
		D	D		
LA	LA	LA	LG	LA	LA

10BF#DLG#
20HCLA 10HCF#DLA 20HCLA

18.

A	A	A	A	A	A
		D	D	D	D
		C	C	C	C
LF#	LF#	LF#	LF#	LF#	LF#

20ALF# 20ADCLF# 20ADCLF#

19.

HE	HE	A	A	A	A
		F#	F#	F#	F#
LA	LA	LA	LA	LA	LA
LLA	LLA	LLA	LLA	LLA	LLA

20HELALLA 20AF#LALLA 20F#LALLA

20.

A	A	A	A	A	A
		D	D	D	D
LF#	LF#	LF#	LF#	LF#	LF#
LLF#	LLF#	LLF#	LLF#	LLF#	LLF#

20ALF#LLF# 20ADLF#LLF# 20ADLF#LLF#

21.

A	A	A	A	A	A
		F#	F#	F#	F#
LD	LD	LD	LD	LD	LD
LLD	LLD	LLD	LLD	LLD	LLD

20ALDLLD 20AF#LDLLD 20AF#LDLLD

22.

HE	HE	HE	HE	HE	HE
		HC	HC	HC	HC
LF#	LF#	LF#	LF#	LF#	LF#
		LLF#	LLF#	LLF#	LLF#

20HELF#LLF# 20HEHCLF#LLF# 20HEHCLF#LLF#

23.

HD	HD	HD	HD	HD	HD
		F#	F#	F#	F#
LA	LA	LA	LA	LA	LA
LLA	LLA	LLA	LLA	LLA	LLA

20HDLALLA 20HDF#LALLA 20HDF#LALLA

24.

HF#	HF#	HF#	HF#	HE	HE
		A	A	F#	F#
LD	LD	LD	LD	LD	LD
LLD	LLD	LLD	LLD	LLD	LLD

20HF#LDLLD 20HF#ALDLLD 20HEF#LDLLD

25.

HD	HD	HD	B	B	B
		B	D	D	D
LG	LG	LG	LG	LG	LG
LLG	LLG	LLG	LLG	LLG	LLG

20HDLGLLG 10HDBLGLLG 10BDLGLLG 20BDLGLLG

26.

G	G	G	G	G	G
D	D	D	D	D	D
LB	LB	LB	LB	LB	LB#
LLD	LLD	LLD	LLD	LLD	LLD

60GDLBLLD

This is how your D.L. LOGO score looks:

```
TO BLOW
         MUSIC [T3600
              20G X
              20BLGLLG X 10BGD X
                10HCGD X 20B
              20GLDLLD X 20DLB X 20G
                X
              20BLGLLG X 10BGD X
                10HCGD X 20B X
              20GLDLLD X 20GDLB 10D X
                10G X
              20BLGLLG X 20BGLGLLG X
                20BGLGLLG X
              20HCLALLA X 20HCDLALLA
                X 20HCDLALLA
              20ALF# X 10ADCLF#
                10G#DCE# X 20AF# X
              20F#LD X 20F#DCLD 1X
                20HCLD X
              20ALF#LLF# X 20F#CLALF#
                X 20ECLALF# X
              20DLF#LLF# X 20DCLALF#
                X 20DCLALF# X
```

```
                    20GLGLLG X 20GDLBLG X
                      20GDLBLG X
                    20BLGLLG X 20BDLGLLG X
                      20BDLGLLG X
                    20HDLB X 10HDBGLB X
                      10HEBGC X 20HDLB X
                    20BLG X 20BGDLG X 20BLG
                      X
                    20HDLB X 20HDGDLB X
                      20HDGDLB X
                    20HEC X 20HEGEC X
                      20HEGEC X
                    20HCLA X 10HCF#DLA
                      10BF#DLG# X 20HCLA X
                    20ALF# X 20ADCLF# X
                      20ADCLF# X
                    20HELALLA X 20AF#LALLA
                      X 20AF#LALLA X
                    20ALF#LLF# X
                      20ADLF#LLF# X
                      20ADLF#LLF# X
                    20ALDLLD X 20AF#LDLLD X
                      20AF#LDLLD X
                    20HELF#LLF#
                      20HEHCF#LLF#
                      20HEHCF#LLF# X
                    20HDLALLA X 20HDF#LALLA
                      X 20HDF#LALLA X
                    20HF#LDLLD X
                      20HF#ALDLLD X
                      20HEF#LDLLD X
                    20HDLGLLG X 10BDLGLLG X
                      10BDLGLLG X 20BLLGLLG
                      X
                    60GDLBLLD]
               END
```

A Note Entry Shortcut

You can use the ampersand (&) to code scores more easily. You use the ampersand to repeat a note in the same position in a previous note-word. However, the ampersand only represents pitch; you must still indicate the duration of the note. The following example shows Line 7 of ''Blow the Man Down''; first, without ampersands, and then, with ampersands.

```
20HCLALLA X 20HCDLALLA X 20HCDLALLA

20HCLALLA X 20&D&& X 20&&&&
```

Going On

There are, of course, many scores more or less complicated than ''Blow the Man Down.'' To enter musical scores that have more than the 4 voices, you must determine which voices are essential to the music or rearrange the score.

A good policy is to pick out music that matches your expertise. If you know little about music but are interested in some more D.L. LOGO transcriptions, you can find a wide selection of easy-to-play scores in music stores and community libraries. If you have advanced training in music, you can make the necessary changes to complicated pieces.

In either case, experiment. If a particular line gives you trouble, enter it as a separate piece. Then, change it until you get the results you want. Use D.L. LOGO's editing features to insert the final version. Much of the enjoyment of D.L. LOGO comes from knowing there is no right or wrong way to write a procedure. The best way is the way that sounds or looks best to you.

Chapter Summary

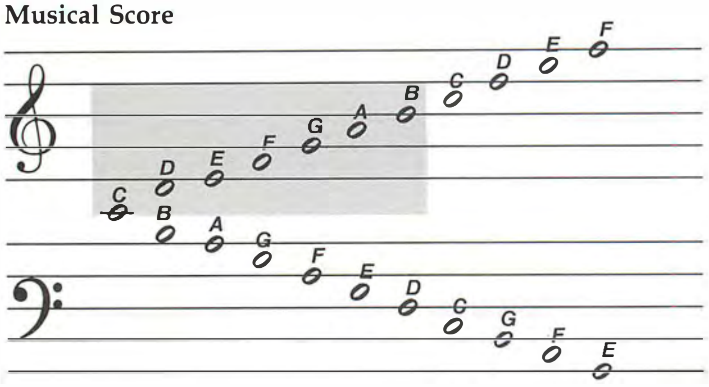
PRIMITIVE	Abbrev.	Purpose
MUSIC	–	Produces musical tones.

Turtle Facts

- Pitch specifications are C, D, E, F, G, A, B, and C, followed by an optional sharp or flat designation (# or ')
- The character *X* indicates a voice is resting
- The ampersand (&) indicates that a pitch remains the same as in the previous setting
- You indicate octave switches with *H* for higher or *L* for lower
- You indicate tempo settings with *T* followed by a value—for example, T120
- D.L. LOGO music can produce 4 voices, simultaneously
- You can use all D.L. LOGO editing procedures with music codes

Now, try D.L. LOGO's many music features with your own pieces. Invite the neighbors over and have a sing-along.

Musical Score



NOTE	VALUE
𝅝	Whole
𝅗𝅥	Half
♩	Quarter
♪	Eighth
𝅘𝅥𝅯	Sixteenth
𝅘𝅥𝅰	Thirty-second

. . . . *About Octaves*

The treble and bass staves at the top of the worksheet have a shaded section that represents the default octave. If the note falls above or below this octave, tell D.L. LOGO by using the letters L *for lower and* H *for higher.*

A note can be several octaves above or below the default octave. In this case, you need to indicate each octave by L *or* H. *For example,* LLLB *indicates the B that is 3 octaves below the default octave. HHD indicates the D that is 2 octaves above the default octave.*

7 VARIETY: THE SPICE OF LIFE

Storing Values in Variables

Section 1	Making Things: Creating words and lists.
Section 2	Handling Data: what a variable is and how you make one.
Section 3	Counting on Variables: using numeric variables.
Section 4	Variables at Home and Abroad: local and global variables.
Section 5	Ins and Outs of LOGO: using OUTPUT in procedures.

Section 1
Making Things: Creating words and lists

LOGO is attractive because it relates to real life. For instance, a *word* in LOGO is exactly what you expect, a *string* of characters. You let LOGO know when a string of characters is a word by placing a quotation mark in front of the first character. For instance, PRINT "HELLO tells LOGO to display the word HELLO.

To indicate the end of a word, press SPACEBAR, TAB, or ENTER. Therefore, the command PRINT "HELLO "FRIEND is acceptable to LOGO but PRINT "HELLO FRIEND is not. Because a space separates HELLO and FRIEND, and FRIEND has no beginning quotation mark, LOGO accepts HELLO as a word but interprets FRIEND as a procedure name.

Making Variables

Typing **PRINT "HELLO** displays the word HELLO only temporarily. If you want to display the word HELLO a number of times, you must store it as a variable. The primitive MAKE creates variables in which you can store words or lists.

A variable resembles a box; it can contain objects, or *values*. For instance, to create a variable (or container) for the word HELLO, type:

```
    ? MAKE "W "HELLO ENTER
```

The letter *W* (which is the variable name) now contains the word HELLO. To test this, type **PRINT :W** ENTER. The screen displays the word HELLO. The colon before the variable name tells LOGO that the following name (in

. . . . About Make

If you are familiar with algebra, the MAKE command is similar to the term let. *While in algebra you say, "let X equal the distance travelled," in LOGO you say "MAKE X equal the distanced travelled." Both* let *and MAKE are used to assign a value to a name.*

. . . . About Colons

In LOGO, colons before variable names are called dots. *Thus, you read the variable :WINTERCLOTHES as* dots WINTERCLOTHES.

*. . . . **About Words***

The following characters can be used in forming words: A-Z, a-z, 0-9, # $ % & @ _ [\ ] ? , . !
Some examples of words are: "HELLO, "$AMOUNT_OWED, 123.456, "#54, "99%, "JACK&JILL, "one_a_day, "Billy_the_Kid

this case W) is a variable. In LOGO, the colon before a variable name is referred to as *dots*, and the preceding variable name is pronounced *dots W*.

Numeric words are an exception to the rule that words must be defined by preceding quotation marks. You can create a numeric word variable, containing the number of days in a year, by typing **MAKE "YEAR 365**. To create a variable containing the number of days in a week, type **MAKE "WEEK 7**. To view the results, type:

```
?PRINT :YEAR :WEEK [ENTER]
365 7
```

Making Quotes

In place of quotation marks ("), you can also use the primitive QUOTE to define a word. For instance, the following 2 command lines produce the same results:

```
PRINT "GREETINGS [ENTER]
PRINT QUOTE GREETINGS [ENTER]
```

Combining Words

You can combine several words into a word variable. When you do so, all the words combine into 1 longer word, without spaces. The following commands show how words can be combined. First, create a new word value for W by typing:

```
? MAKE "W "ONE [ENTER]
```

Display the word by typing:

```
? :W[ENTER]
ONE
```

To create a longer word, type:

```
? MAKE "COMBO WORD :W "TWO "THREE
"FOUR ENTER
? :COMBO ENTER
ONETWOTHREEFOUR
```

The WORD primitive combines the variable W with the words TWO, THREE, and FOUR and creates the word ONETWOTHREEFOUR. You can use WORD at any time to make 1 word of several words or word variables.

Types of Variables

Variables can be either data variables or numeric variables. Either of these can be words or lists. A quotation mark defines a word variable, and square brackets define a list variable. For instance, to create a data list of your winter clothes, type:

```
? MAKE "WINTERCLOTHES [PARKA MITTS
BOOTS SOCKS TOQUE
LONGJOHNS MUFFS] ENTER
```

To create a numeric word variable that represents the number of clothes in the WINTERCLOTHES variable, type:

```
? MAKE "NUMBERCLOTHES 7 ENTER
```

These 2 commands make the WINTERCLOTHES variable into a list and the NUMBERCLOTHES variable into a word. You can see the kind and number of clothes at any time by typing:

```
? PRINT :WINTERCLOTHES
:NUMBERCLOTHES ENTER
PARKA MITTS BOOTS SOCKS TOQUE
  LONGJOHNS MUFFS 7
```

. . . About Variables

Think of variables as containers that have no fixed value or contents. A suitcase is a variable container; it can hold your clothes or your friend's clothes or your aunt's clothes, all in varying amounts. The same is true with a LOGO variable. You can put any value you like in a variable and give it any name you like. A variable named CUP can have a value of WATER, or APPLE JUICE, or ROOT BEER, or any combination of the 3.

Although variables can change, it is not always necessary. CUP can contain the value WATER throughout the course of a program. The fact that it is a variable means that it can change.

. . . . About Lists

A list is defined by enclosing words or other lists within square brackets. Words within a list do not require leading quotation marks. Some examples of lists are:

```
[HOW ARE YOU]
[YOUR BILL IS
 $44.90]
[101 102 103 104]
[[10 TO THE POWER
 OF 2][IS 100]]
```

Empty Boxes

You can also have empty variables. An empty variable is available for storage, but has nothing in it. To create an empty word, use a quotation mark without any following characters, such as:

```
MAKE "BOOK " ENTER
```

To create an empty list, use 2 square brackets without characters between them, such as:

```
MAKE "BOX []
```

It may seem odd to create empty variables, but they can be useful tools. When you PRINT an empty variable, the screen displays a blank line as shown below:

```
? PRINT :BOOK ENTER

?
```

To see how an empty list variable is stored, use the SHOW primitive. Type:

```
? SHOW :BOX ENTER
[]
```

SHOW displays a variable with all its associated brackets.

Section 2
Handling Data

Lists Inside Lists

You already have a list of your winter clothes neatly stored in the variable WINTERCLOTHES. Use the same procedure to make a record of all your clothes:

```
? MAKE "SUMMERCLOTHES [SHIRTS [SWIM
SUIT] JOGGERS SHORTS] ENTER
? MAKE "SPRINGCLOTHES [JACKET
RAINCOAT BOOTS SWEATER] ENTER
```

Notice that SUMMERCLOTHES contains a list within a list. Use separate brackets to enclose SWIM SUIT and prevent LOGO from treating SWIM and SUIT as 2 separate elements in the list.

To take inventory of all your clothes without typing WINTERCLOTHES, SUMMERCLOTHES and SPRINGCLOTHES separately, you can combine the 3 lists:

```
? MAKE "CLOTHES SE :WINTERCLOTHES
:SUMMERCLOTHES :SPRINGCLOTHES ENTER
```

In this case the primitive SENTENCE (abbreviated SE) precedes the 3 lists to indicate that they are 1 long list. Otherwise, LOGO does not include the other 2 lists in the new variable. To see the value of CLOTHES, type:

```
? :CLOTHES ENTER
[PARKA MITTS BOOTS SOCKS TOQUE
LONGJOHNS MUFFS SHIRTS [SWIM SUIT]
JOGGERS SHORTS JACKET RAINCOAT BOOTS
SWEATER]
```

The colon (:) before a name tells LOGO you are referring to a variable, not a procedure name. Using *dots* before a variable name causes LOGO to produce the contents of the variable. Because dots only precede variables, you can use primitive names as variable names.

Instead of SENTENCE, you can also use the primitive LIST to combine lists, but the result is different. To see the difference, use the LIST primitive to create another CLOTHES list:

```
? MAKE "CLOTHES LIST :WINTERCLOTHES
:SUMMERCLOTHES :SPRINGCLOTHES ENTER
? :CLOTHES ENTER
[[PARKA MITTS BOOTS SOCKS TOQUE
LONGJOHNS MUFFS] [SHIRTS
[SWIM SUIT] JOGGERS SHORTS] [JACKET
RAINCOAT BOOTS
SWEATER]]
```

Using the SENTENCE primitive strips 1 level of brackets from the final list, but LIST leaves all brackets intact.

Another way to produce the contents of a variable is to use the primitive THING. Using THING has the same effect as preceding a variable with dots. For instance:

```
? MAKE "JAR "COOKIES
? PRINT :JAR ENTER
COOKIES
? PRINT THING "JAR ENTER
COOKIES
```

THING can also display an additional variable level. For instance, create a third level by typing **MAKE "COOKIES "GRANDMAS**. Now try the following:

```
? PRINT THING :JAR
GRANDMAS
```

By using both THING and dots, the screen displays GRANDMAS rather than COOKIES.

You have already been introduced to the SHOW primitive. SHOW's main function is to aid you in uncovering problems in procedures. When you wish to display the contents of a variable, you normally use PRINT. But, if the contents do not display correctly, use SHOW to display the complete variable. For example, suppose you wished to display a variable list named DOGS, as:

```
DOBERMAN
GREAT DANE
HUSKY
```

However, when you display the list, it shows:

```
DOBERMAN
GREAT
DANE
HUSKY
```

To see what is wrong, use the SHOW primitive. Type:

```
? SHOW :DOGS ENTER
[DOBERMAN GREAT DANE HUSKY]
```

You can see that GREAT DANE is two separate elements in the list. To correct the problem, change the DOGS list by typing:

```
MAKE "DOGS [DOBERMAN [GREAT DANE]
HUSKY] ENTER
```

Linking Variables

You can link variables into chains of any length to keep track of related data. For instance, create a variable by typing:

```
? MAKE "WALLET "MONEY ENTER
```

Now, every time you type :WALLET, your screen displays MONEY. You can interpret this linking several ways:

```
. WALLET is the name of MONEY
. MONEY is the value of WALLET
. MONEY is the thing of WALLET
. MONEY is the contents of WALLET
```

As you study the LOGO language, you may see variables described in any of these ways.

Create a longer chain by typing:

```
? MAKE "MONEY "12.45 ENTER
? MAKE "12.45 "MINE ENTER
```

To review the chain you have created, type:

```
? :WALLET ENTER
MONEY
? :MONEY ENTER
12.45
? :12.45 ENTER
MINE
```

You can also use the primitive THING? to discover whether a variable has an object value. To see if the WALLET has an object linked to it, type:

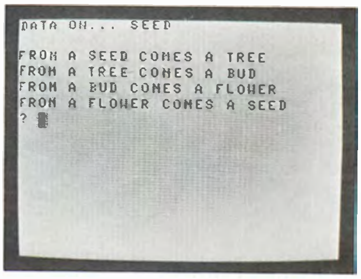
```
? THING? :WALLET ENTER
TRUE
```

Data in Storage

The following program uses the linking concept to store related data:

```
TO LINK
     PRINT1 [ENTER THE OBJECT
      NAME...]
```

```
        INPUT
        MAKE "NAME :I
        REPEAT 3 [
              MAKE "J :I
              PRINT1 :I [\ LINKS TO..]
              INPUT
              MAKE THING "J :I]
        MAKE THING "I :NAME
        UNLINK :NAME
  END

  TO INPUT
        MAKE "I "
        WHILE NOT MEMBER? CHAR 13 :I
              [MAKE "I WORD :I RC
              PRINT1 LAST :I]
        MAKE "I BUTLAST :I
  END

  TO UNLINK :NAME
        CLEARTEXT
        PRINT [DATA ON...] :NAME
        PRINT
        MAKE "I :NAME
        REPEAT 4 [
              PRINT1 [FROM A\ ] :I
               [\ COMES A\ ]
              MAKE "I THING :I
              PRINT :I]
  END
```



```
DATA ON... SEED

FROM A SEED COMES A TREE
FROM A TREE COMES A BUD
FROM A BUD COMES A FLOWER
FROM A FLOWER COMES A SEED
?
```

Many features of this program may not be familiar to you at this time. You learn about these new commands and features in later chapters. For now, the following information shows you what the 2 procedures accomplish in linking and displaying linked variables.

A sample session of the program might look like this (user input appears in lowercase to distinguish it from the program display):

```
ENTER THE OBJECT NAME...seed ENTER
SEED LINKS TO..plant ENTER
PLANT LINKS TO..bud ENTER
BUD LINKS TO..flower ENTER

DATA ON... SEED

FROM A SEED COMES A PLANT
FROM A PLANT COMES A BUD
FROM A BUD COMES A FLOWER
FROM A FLOWER COMES A SEED
```

Enter and execute the program. Using the CONTENTS primitive, you can see how the data you input is linked:

```
? CONTENTS ENTER
FLOWER=SEED
BUD=FLOWER
PLANT=BUD
SEED=PLANT
J=BUD
NAME=SEED
I=SEED
```

Except for the last 3 variables on the list, you can see that each item entered is linked to the next item, like elephants walking in single file, holding each other's tails. By knowing the name of the initial variable (in this case, SEED), you can extract all other data.

Section 3
Counting on Variables

The most common use of variables is in arithmetic operations, and especially in Turtle graphics routines. For instance, suppose you want to create a number of squares of different sizes. The procedure to produce 1 square might look like this:

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

To draw squares of different scales, replace the number 50 with a variable name; for instance, STEPS. To select the size of the square, you must change the procedure to include a variable input. Change it to look like this:

```
TO BOX :STEPS
      REPEAT 4 [FD :STEPS RT 90]
END
```

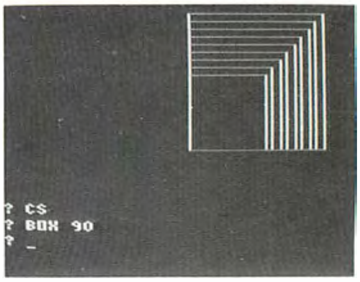
When you include a variable as part of a procedure name, you must precede the variable with dots. Then, when you execute the procedure, you must include an argument, or input value, for each variable included in the name. For instance, to execute the preceding procedure, type:

```
? BOX 90 ENTER
```

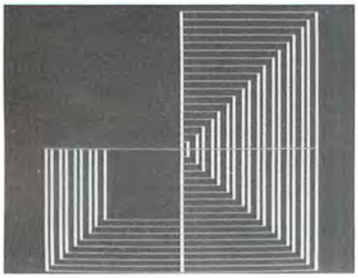
The variable STEPS now contains a value of 90, and the BOX procedure draws a square with sides 90 steps long.

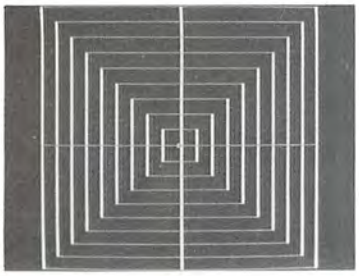
Create another square by typing:

```
? BOX 50 ENTER
```



```
? CS
? BOX 90
? _
```





This time the Turtle draws a square with sides 50 steps long.

You can display any number and size of squares. To reduce the size of the square at a set rate, manipulate STEPS by decreasing the variable each time the Turtle completes a square. The following program does this:

```
TO BOX :STEPS
     REPEAT 9 [
       REPEAT 4 [
         FD :STEPS RT 90]
       MAKE "STEPS :STEPS - 5]
END
```

Now, the value you give to STEPS decreases 5 each time 1 of 9 squares is drawn.

More Appeal

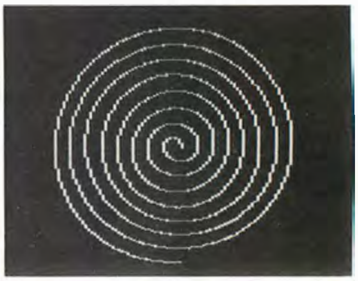
Make your BOX procedure more attractive by editing it to match the following:

```
TO BOX
   CS
   FULLSCREEN
   REPEAT 4 [MAKE "STEPS "90
       REPEAT 18 [REPEAT 4
           [FD :STEPS RT 90]
           MAKE "STEPS :STEPS - 5]
           RT 90]
END
```

More Varying Variables

Suppose you are creating a program that uses numerous right turn commands. You can either create separate command lines for each turn, or you can use 1 turn command with a variable that specifies the degree of turn.

The following spiral procedure illustrates another way to use variables. To create a spiral, you must constantly change the distance traveled or the degree of turn or both. This procedure creates a spiral by decreasing the amount of forward movement before each turn. By changing the value of TURN and MOVE, you can alter the distance between the lines of the spiral.



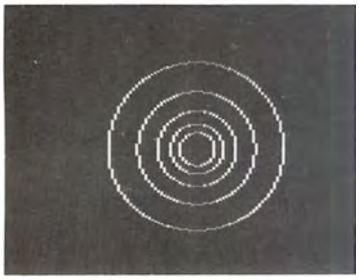
```
TO SPIRAL
     CS
     FULLSCREEN
     SETY -80 LT 90
     MAKE "TURN 2
     MAKE "MOVE 3
     REPEAT 14 [
       MAKE "MOVE :MOVE-.1*:TURN
         REPEAT 100
           [FD :MOVE RT :TURN]]
END
```

Going in Circles

A similar method creates concentric circles by using the variables TURN and JUMP. TURN is the number of degrees that the Turtle turns after each forward step. JUMP is the distance from the center of the screen that the Turtle jumps before it draws the next circle.

In the following procedure, TURN increases 1.5 times its former value to create progressively smaller circles. JUMP, which is initially set to 5Ø, is decreased by 29 and divided by the value of TURN, after each circle is drawn. Consequently, the beginning of each circle is closer to the center of the screen.

Explanations for some primitives in these programs that you do not yet recognize appear later in the manual. For now, study the operations of the variables to see how



you can use them to reduce the number of commands needed to complete a particular task.

```
TO BULL
      CS
      MAKE "JUMP 50
      HOME
      MAKE "TURN 1
      REPEAT 5 [
            HOME
            PU LT 90 FD :JUMP RT 90
             PD
            CIRCLE
            MAKE "TURN :TURN +
             .5*:TURN
            MAKE "JUMP :JUMP-29/:TURN
]
END

TO CIRCLE
      REPEAT 360/:TURN [
        FD 1 RT :TURN ]
END
```

Section 4 Variables at Home and Abroad

Most of the variables discussed in the preceding sections are global variables; they automatically pass from procedure to procedure. There are also *local variables* that are restricted to the procedure that created them or to any procedure called by the procedure that created them.

With local variables, you can use the same variable name in more than 1 procedure without causing conflicts. To illustrate this capacity, type the following procedure:

```
TO VARI :T
        IF :T>100 [STOP]
        PRINT :T
        MAKE "T :T+10
        VARI :T
END
```

Execute it from the immediate mode by typing VARI and a number, such as **VARI 10** ENTER. Type **CONTENTS** ENTER after the program ends to see that D.L. LOGO did not store any global values, even though you give T an initial value. This happens because you lose the values of local variables once a procedure ends. Any variable input provided as an argument (transferred through the use of a variable in a procedure name) is *local* to the procedure that receives it.

Notice that the line before END is VARI :T. This command causes the procedure to reexecute itself until the value of T exceeds 100. Every time the program executes itself, it establishes a new local variable T. To see that the variable T is indeed created numerous times to store multiple values, add 1 more line to the procedure:

. . . . Arguments in Procedure Names

LOGO uses variable names in procedure names to pass data between procedures. The following example shows how this is done:

```
TO MOTOR :DISTANCE
```

When you execute the MOTOR procedure, or it is called by another procedure, a value for DISTANCE must be provided, such as MOTOR 50. An example of such a procedure is:

```
TO MOTOR :DISTANCE
    MAKE "GALLONS
     34
    MAKE
     "CONSUMPTION
     :DISTANCE/
     :GALLONS
    PRINT [YOU
     USED]
     :CONSUMPTION
     [GALLONS OF
     GASOLINE]
END
```

To create a procedure that accepts more than one argument, separate the selected variable names by a space, for instance:

```
TO GO :DIST :GAS
```

```
TO VARI :T
      IF :T>100 [STOP]
      PRINT :T
      MAKE "T :T+10
      VARI :T CONTENTS
END
```

Now LOGO displays the contents of all variables before the procedure concludes. In this case, T has 55 separate values that LOGO keeps in perspective.

You can also establish local variables by defining them with the primitive LOCAL. Once you define a variable as a local, it can be used by only the procedure in which it is created, or by a procedure called by the *parent* procedure. However, the same procedure name can also be used in a *higher* procedure. The two variables have no relationship, even though they bear the same name. Test the LOCAL primitive with this procedure:

```
TO HIGHER
      MAKE "T 55
      VARI
END

TO VARI
      LOCAL "T
      MAKE "T 10
      MAKE "S 10
END
```

After you execute the program, type **CONTENTS** ENTER. Although T was defined twice as a variable name, it has only 1 value, the global value set in the first procedure. The local variable T, created in the VARI procedure, has no effect on the global variable T and loses its value when VARI ends. The variable S, that was also created in the VARI procedure as a global variable, retains its value after the procedure ends.

Section 5
The Ins and Outs of LOGO

Putting Out

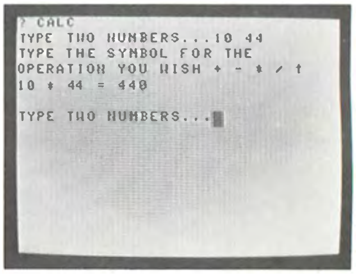
In some instances, the purpose of a primitive is to produce an output. Such is the case with XCOR and YCOR, which output the current Turtle's location. Many of the procedures you write produce input to other procedures.

So far in this manual, procedures have used variables to transfer data between procedures. The OUTPUT primitive transfers values without using variables. The following procedure uses OUTPUT in a calculator program.

```
TO CALC
      PRINT1 [TYPE TWO NUMBERS:]
      MAKE "N RQ
      MAKE "N1 FIRST :N
      MAKE "N2 LAST :N
      PRINT [TYPE THE SYMBOL FOR THE]
      PRINT [OPERATION YOU WISH\ -
       \+\*\↑ \/ ]
      MAKE "S RC
      PRINT :N1 :S :N2 "= GIVE
      PRINT CALC
END

TO GIVE
      SELECT [

          :S = "+ [OUTPUT :N1 + :N2]
          :S = "- [OUTPUT :N1 - :N2]
          :S = "* [OUTPUT :N1 * :N2]
          :S = "↑ [OUTPUT :N1 ↑ :N2]
          :S = "/ [OUTPUT :N1 / :N2]]
END
```

. . . . About Special Characters

All LOGO arithmetic and comparative operators are considered special characters by LOGO, and are treated differently than other characters. To include an arithmetic or comparative operator in a word, you must precede it with a backslash (MAKE "W "2\ +3), or use the WORD primitive (MAKE "W WORD "2 "+ "3). To include an arithmetic or comparative operator in a list, use a backslash (MAKE "L [2\ +3]). Chapter 9 contains a list of all arithmetic and comparative operators.



```
? CALC
TYPE TWO NUMBERS...10 44
TYPE THE SYMBOL FOR THE
OPERATION YOU WISH + - * / ↑
10 * 44 = 440

TYPE TWO NUMBERS...
```

Line 9 uses *output* from the procedure GIVE as the answer to a calculation. In effect, OUTPUT lets you use a procedure as though it were a variable. The procedure GIVE is used in place of a number or variable value for the PRINT primitive. To understand this better, try the following example:

```
TO DEMO
        PRINT B
END

TO B
        OUTPUT "HELLO
END
```

In this example, the procedure B outputs a value, or an argument, for the PRINT primitive in DEMO, and the word HELLO is displayed.

Recursive Outputs

As previously demonstrated in the VARI procedure, procedures can output values to themselves. When this occurs, the procedure is called *recursive.* A procedure that calculates factorials is a good example of recursion.

The factorial of a number is the product of all positive integers from 1 to that number. For instance, the factorial of 4 is $1\times2\times3\times4=24$. To find the factorial of a large number, say 51, you need not type each number from 1 to 51. Instead, you can use a procedure such as the following:

```
TO FACT :N
        IF :N = 1 [OUTPUT :N]
        OUTPUT :N*FACT :N-1
END
```

. . . . *About Recursion*

"A computer is a machine that computes." This is not a very good definition for a dictionary because the definition is circular *— it repeats itself. However, in Logo, such a repetitive process is not only acceptable, but is an excellent programming device. Causing a procedure to use itself is called recursion, and recursion is a very powerful Logo function.*

. . . . *About Input Variables*

Procedures can be defined to include any number of input variables. You do this by adding variable names to the procedure name, such as:

```
TO ADD :A :B :C :D
        MAKE "TOTAL :A
        + :B + :C + :D
END
```

To use the program type:

```
? ADD 5 10 15 20
50
```

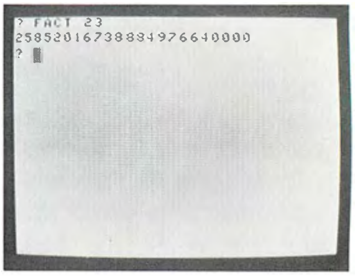
The asterisk is used in LOGO to indicate multiplication. The variable N is multiplied by the next result of the FACT procedure. To use this procedure for finding the factorial of 4, type:

```
? FACT 4 ENTER
24
```

Using OUTPUT lets you recycle the result of each operation through the procedure again and again, decreasing the *local* variable :N until it reaches 1. The value of the *global* variable :N (the completed factorial) then appears as output on the screen.

Using this procedure, you can find the factorial of 51 in approximately 2 seconds. Calculating the factorial of 100, a 158 digit number, takes about 8 seconds.

Recursive procedures and the OUTPUT primitive can provide fascinating opportunities for experimentation. Look for more information in Chapter 10.



```
? FACT 23
25852016738884976640000
? 
```

Chapter Summary

PRIMITIVE	Abbrev.	Purpose
MAKE	–	Establishes variables and their values.
QUOTE	–	Defines a word.
THING	–	Indicates that a word is a variable name. The same as a colon or dots.
LOCAL	–	Establishes a variable as a local.
OUTPUT		Terminates the action of a procedure and returns ts data to the calling procedure.

Turtle Facts

- A variable is the name or label you give to an object.
- An object is a word, list, or sentence.
- Create global variables with the primitive MAKE.
- A local variable is only valid in the procedure that creates it or in a procedure called by the originating procedure.
- A global variable can pass between any number of procedures without losing its value. It also retains its value after a program concludes.
- You create local variables when you pass arguments, or inputs, through a procedure name or when you use

the LOCAL primitive. A local variable cannot be passed to a higher procedure, and its value is lost when its originating procedure ends.

- You can combine several variables into 1 variable.

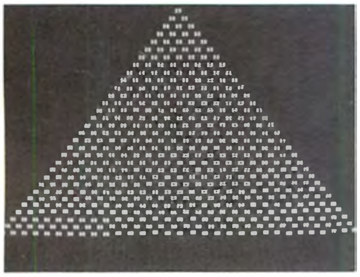
- Variables can link, such as: ONE = TWO, TWO = THREE, THREE = FOUR.

- When using a variable in LOGO, a colon (:), called *dots*, must precede the variable.

- All arithmetic and comparative operations function with variables.

Suggested Project

Using the manipulation of variables and primitives, write a program that stacks blocks in a pyramid fashion, similar to the accompanying illustration. Use the SETX, SETY, XCOR, and YCOR primitives.



Possible Solution

```
TO STACK
      MAKE "A 0
      CS
      SETXY 0 90
      BOX
      SETY YCOR-5
      REPEAT 30 [
          MAKE "X XCOR-4
          MAKE "A :A + 4
          FOR "I :X :A 8
              [SETX :I BOX]
          SETXY :X YCOR-5]
END

TO BOX
      REPEAT 4
      [FD 2 RT 90]
END
```

8 TURTLE'S JUGGLING ACT

Handling Text, Editing, Time

Section 1	Words, Numbers, and Lists: how to handle them.
Section 2	Making a List: the difference between lists and words.
Section 3	Talking in Sentences: what you can do with a sentence.
Section 4	The First Shall Be Last: how to manipulate words, lists, and sentences.
Section 5	Running Away With Lists: using RUN to execute lists.
Section 6	Turtle Swallowed a Clock: using the built-in date and time function.

Section 1
Words, Numbers, and Lists

The ability to handle words and lists is the basis for LOGO's power. This section shows you how to juggle variables and their contents to create new variables or rearrange old ones.

Adding Numbers

LOGO treats numbers as words, but quotation marks are not required. For instance, the following 3 examples produce the same result.

```
? MAKE "N WORD "1 "2 "3 "4 ENTER
? MAKE "N (WORD 1 2 3 4) ENTER
? MAKE "N WORD 1234 ENTER
```

You can also use arithmetic operators with words. For example:

```
? PRINT (WORD 1 + 3) ENTER
4
```

Similarly, the following 3 commands produce the same sum of 24.

```
? PRINT "12 + "12 ENTER
? PRINT 12 + 12 ENTER
? PRINT 12+12 ENTER
```

To display the numbers and the arithmetic sign without doing the addition, use the following command:

```
? PRINT "12 "+ "12 ENTER
12 + 12
```

It isn't possible to list all the ways in which you can use numbers as words. Experiment and see how numeric

. . . . About Words

There are two kinds of words in D.L. LOGO: procedure names and objects. Procedure names are words that tell D.L. LOGO to perform an action, primitives or procedure names. Objects are words that D.L. LOGO doesn't know how to do. *They are just* things *that Logo can manipulate in different ways, such as print or combine with other words.*

. . . . About Parentheses

You use parentheses in command statements to clarify the order of operations. The procedure, as well as the argument, must be enclosed within the parentheses. Following is an example of a statement with, and without parentheses:

MAKE "R (RANDOM 11)+1
MAKE "R RANDOM 11+1

The first statement creates a random number in the range 1-11. The second statement creates a random number in the range 0-11.

words behave under various command syntax. Some possible combinations you can try are:

```
PRINT WORD 1 2 3 + (WORD 1)
PRINT (WORD 1 2 3) + (WORD 1 2 3)
PRINT (WORD 1 2 3) / 2
PRINT WORD 1 2 3 + WORD 1 2 3
PRINT WORD 1 + 1 + 1 + 1 / 2
PRINT (WORD 1 + 1 + 1 + 1) / 2
```

Section 2
Making a List

It is important that you understand the difference between words and lists and that you know how to recognize each. Several LOGO primitives operate only on words, and several operate only on lists. Often, the results of operations on words and lists look the same. For example, type the following 2 command lines, and type **JOHN** in response to the RQ prompts:

```
? MAKE "INPUT1 RQ [ENTER]
JOHN
? MAKE "INPUT2 FIRST RQ [ENTER]
JOHN
```

To observe the results, type:

```
? :INPUT1 [ENTER]
[JOHN]
? :INPUT2 [ENTER]
JOHN
```

The contents of INPUT1 is displayed with square brackets because it is a list, but the contents of INPUT2 is a word and is not enclosed in brackets. However, if you use PRINT to display the contents of the two variables, the values will appear identical:

```
? PRINT :INPUT1 [ENTER]
JOHN
? PRINT :INPUT2 [ENTER]
JOHN
```

Another way to check whether the value stored in a variable is a word or a list is with SHOW, as described in previous chapters. For example:

```
? SHOW :INPUT1 [ENTER]
[JOHN]
```

*. . . . **About SHOW***

LOGO recognizes the primitive SHOW by displaying the specified variable or data on the screen. In the immediate mode, however, SHOW is always implied. For instance:

SHOW "HELLO
and
"HELLO

both produce the same result: they display the word HELLO. Similarly, both SHOW "L and :L display the the variable L on the screen.

```
? SHOW :INPUT2 ENTER
JOHN
```

The primitives LIST? and WORD? also test lists and words. Type:

```
? PRINT WORD? :INPUT1 ENTER
FALSE
? PRINT LIST? :INPUT1 ENTER
TRUE
? PRINT WORD? :INPUT2 ENTER
TRUE
? PRINT LIST? :INPUT2 ENTER
FALSE
```

If the variable is a word, WORD? displays TRUE. If it is not a word, FALSE is displayed. LIST? works in the same manner.

Section 3 Talking In Sentences

The SENTENCE primitive assembles words and lists into single lists. For instance, type the following:

```
? MAKE "LINE1 [ROSES ARE] ENTER
? MAKE "LINE2 [VIOLETS ARE] ENTER
? MAKE "LINE3 [IS SWEET] ENTER
? MAKE "LINE4 [AND SO ARE] ENTER
? MAKE "POEM SENTENCE :LINE1 "RED
  :LINE2 [BLUE SUGAR] :LINE3
  :LINE4 "YOU ENTER

? :POEM ENTER
```

The screen shows:

```
[ROSES ARE RED VIOLETS ARE BLUE
 SUGAR IS SWEET AND SO ARE YOU]
```

To see the result created when LIST is used with the same words and lists, type:

```
? MAKE "POEM1 LIST :LINE1 "RED
  :LINE2 [BLUE SUGAR] :LINE3
  :LINE4 "YOU ENTER
? :POEM1 ENTER
[[ROSES ARE] RED [VIOLETS ARE]
 [BLUE SUGAR] [IS SWEET] [AND SO
 ARE] YOU]
```

Because SENTENCE removes one level of brackets, it combines all the members into 1 long list; LIST keeps the original groupings of the various members.

. . . . About Sentences, Words, and Lists

This manual refers to the contents of sentences, words, and lists as members or elements. For instance, the members of the word HELLO are the characters H, E, L, L, and O. The members of the list [ONE TWO THREE] are the words ONE, TWO, and THREE.

Section 4
The First Shall Be Last

Now that you know how to make words, lists, and sentences, you are ready to see the power of the LOGO primitives that manipulate data.

The key primitives for manipulating the contents of words and lists are:

PRIMITIVE	FUNCTION
FIRST	extracts the first element of a variable.
LAST	extracts the last element of a variable.
BUTFIRST	extracts all but the first element of a variable.
BUTLAST	extracts all but the last element of a variable.
FPUT	adds a specified element to the front of a variable.
LPUT	adds a specified element to the end of a variable.
ITEM	extracts a specified element from a variable.
MEMBER?	tests whether a specified element is included in a variable.
PIECE	extracts a specified number of elements from a variable.

These primitives operate on words, lists, or sentences to extract, insert, count, or pinpoint data. You can select a character or word by its position in a word, list, or sentence. For instance, to see how you might use this technique for a countdown, type the following program:

```
    TO COUNTDOWN
             MAKE "WORD1 123456789
             LABEL "COUNTING
             MAKE "COUNT LAST :WORD1
             MAKE "WORD1 BUTLAST :WORD1
             PRINT :COUNT
             IF :COUNT =1 [STOP]
             GO "COUNTING
    END
```

This program operates as follows:

- Line 2 establishes WORD1 as the numeric sequence *123456789*.
- Line 4 uses the LAST primitive to set the variable COUNT equal to the position of the last element of WORD1.
- Line 5 uses the BUTLAST primitive to redefine the value of WORD1 as all of WORD1, minus the last element.
- Line 6 displays the value of COUNT on the screen.
- Line 7 tests to see if the countdown reached the last element (*1*), and stops the procedure if it did.
- Line 8 loops the program back to get the next element in the word.

The countdown does not include the number 10 because extracting a 2-digit number from a word requires further steps. If you write the procedure using a list rather than a word, including the 10 is easy, as shown in the following procedure. This time the procedure counts from first to last, rather than last to first:

```
    TO COUNTDOWN
             MAKE "LIST1 [10 9 8 7 6 5 4
              3 2 1]
```

*. . . . **About Spaces***

*Because spaces mark the end of a word, you cannot include them in a word. The command: **PRINT (WORD "HI " "THERE)** results in the contents of :W being HITHERE rather than HI THERE. You can, however, force spaces into words by using the backslash character (\) before a space. For instance, the following example displays a word with a space:*

? PRINT "HI\ THERE
HI THERE

```
                 LABEL "COUNTING
                 MAKE "COUNT FIRST :LIST1
                 MAKE "LIST1 BUTFIRST :LIST1
                 PRINT :COUNT
                 IF :COUNT =1 [STOP]
                 GO "COUNTING
END
```

There are times when spaces do not sufficiently divide the members of a list, such as *LETTUCE, CORN, PEAS, GREEN BEANS, CARROTS, HORSE RADISH*. Fortunately, LOGO lets you place lists within lists. To demonstrate this, type the following:

```
MAKE "LIST1 [LETTUCE CORN PEAS [
 GREEN BEANS] CARROTS [HORSE
 RADISH]] ENTER
```

You can see how these lists within lists are handled by changing the COUNTDOWN procedure to match the following:

```
TO COUNTDOWN
                 MAKE "LIST1 [LETTUCE CORN
                  PEAS [GREEN BEANS] CARROTS
                  [HORSE RADISH]]
                 LABEL "COUNTING
                 IF EMPTY? :LIST1 [STOP]
                 MAKE "COUNT FIRST :LIST1
                 MAKE "LIST1 BUTFIRST :LIST1
                 PRINT :COUNT
                 GO "COUNTING
END
```

This procedure introduces the EMPTY? primitive in Line 5. EMPTY? determines when a list or word contains 0 elements.

FIRST, BUTFIRST, LAST, and BUTLAST can be combined in a command. To demonstrate this, type the following command:

```
? PRINT FIRST BUTFIRST [ONE TWO
THREE FOUR FIVE] ENTER
```

LOGO displays the word TWO. To follow the command's logic, read the primitives in the command from right to left. BUTFIRST refers to all of the list except the first element. FIRST refers to the first element of the remainder, which is TWO.

Even the command:

```
? PRINT FIRST BUTFIRST BUTFIRST
BUTFIRST BUTFIRST [ONE TWO THREE
FOUR FIVE] ENTER
```

is valid and produces the word FIVE. Of course it is easier to issue a LAST primitive to accomplish the same thing.

Adding On

The primitives FPUT and LPUT let you easily add data to lists. FPUT inserts an item at the beginning (first) of a list. LPUT inserts an item at the end (last) of a list. These insertions, however, only temporarily append the new item to the list. To append them permanently, use the MAKE primitive.

Create the following shopping list by typing:

```
? MAKE "SLIST [BACON EGGS MILK BREAD
BUTTER] ENTER
```

To display HONEY at the end of the list:

```
? LPUT "HONEY :SLIST ENTER
```

The screen shows:

```
[BACON EGGS MILK BREAD BUTTER HONEY]
```

However, HONEY is not permanent unless you type a command like:

```
? MAKE "SLIST LPUT "HONEY :SLIST
ENTER
```

If this looks confusing, you might find the command easier to understand if you use parentheses, such as this:

```
? MAKE "SLIST (LPUT "HONEY :SLIST)
```

To add HONEY to the front of the list, you keep the command syntax exactly the same, except you type FPUT instead of LPUT.

Item by Item

As noted earlier, you can use multiple BUTFIRST and BUTLAST commands to access members buried in a list, a sentence, or a word. When more than 3 or 4 members are involved, this procedure becomes awkward. Remember the following command line?

```
PRINT FIRST BUTFIRST BUTFIRST
BUTFIRST BUTFIRST [ONE TWO THREE
FOUR FIVE]
```

The ITEM primitive provides a way to extract members with greater ease. The ITEM format is:

Parameter 1 —	the number of the element to extract
Parameter 2 —	the object (word, list, or sentence) from which you wish to extract an element

To display the fifth element or character from the word ABCDEFG, type:

```
? ITEM 5 "ABDEFG ENTER
F
```

To display the third element from the list ONE TWO THREE FOUR FIVE, type:

```
? ITEM 3 [ONE TWO THREE FOUR FIVE]
ENTER
THREE
```

The COUNT

Use the primitive COUNT to find the length of a word, list, or sentence. COUNT returns the number of members in an object. For instance, type:

```
? COUNT [ONE TWO THREE FOUR
FIVE] ENTER
5
```

or

```
? COUNT "12345678 ENTER
8
```

The first instance counts the members of the defined list and returns the number 5. The second instance counts the members of the defined word and displays 8.

COUNT can be used to display any item in a list by referencing its relative position (its position relative to the first or last items). Suppose that the variable LOT contains the list [ONE TWO THREE FOUR FIVE SIX]. To display the third from the last item, type:

```
? ITEM (COUNT :LOT)-2 :LOT
```

The screen displays FOUR.

A PIECE of the Action

The PIECE primitive provides a function similar to ITEM, but lets you extract any number of members from an object. The format for PIECE is:

Parameter 1 — the number of the first element to extract
Parameter 2 — the number of the end element to extract
Parameter 3 — the list from which you wish to extract

To extract element 5 from the list [ONE TWO THREE FOUR FIVE SIX], type:

```
? PRINT PIECE 5 5 [ONE TWO THREE
FOUR FIVE SIX] ENTER
```

FIVE appears on the screen. If you wish to extract both the fourth and fifth members, type:

```
? PRINT PIECE 4 5 [ONE TWO THREE
FOUR FIVE SIX] ENTER
```

The screen displays FOUR and FIVE.

With PIECE you can create new lists or sentences from other lists or sentences. For instance, to create a new list named EXTRACT, type:

```
? MAKE "EXTRACT PIECE 4 5 [ONE TWO
THREE FOUR FIVE SIX] ENTER
```

The variable EXTRACT now contains FOUR FIVE.

PIECE operates in the same manner to extract members of words. To extract the third through seventh elements of the word MISTAKEN, type:

```
? PRINT PIECE 3 7 "MISTAKEN
STAKE
```

Calling Roll

To test whether a particular element is part of a word, list, or sentence, use the MEMBER? primitive. For instance, if you have the list [ONE TWO THREE FOUR FIVE SIX] named LOT, and want to see if that list contains the element FOUR, type:

```
? MEMBER? "FOUR :LOT ENTER
```

LOGO displays the word TRUE to let you know that FOUR is in the list. If you type:

```
? MEMBER? "SEVEN :LOT ENTER
```

LOGO displays the word FALSE. You can use the same command syntax with words and sentences.

WHERE is the Action?

Once you know whether or not an element exists in an object, you can find where it exists. WHERE, used with MEMBER?, pinpoints the position of an element. For instance, use the same list as in the previous example, but type:

```
? MEMBER? "FOUR :LOT
TRUE
? WHERE ENTER
4
```

MEMBER? tells you if an element exists as an element of an object, and WHERE pinpoints the location of the specified element. You can use WHERE with logic operations, for instance:

```
? MEMBER? "FOUR :LOT ENTER
TRUE
```

```
? IF WHERE=4 [PRINT [FOUR IS WHERE
IT SHOULD BE]] ENTER
FOUR IS WHERE IT SHOULD BE
```

You learn more about logic operations, including IF, in the next chapter.

The following 2 procedures demonstrate list manipulation in action. The first procedure deletes a specified element from the list named LOT. The second procedure inserts a new element at a specified location in the list named LOT.

Delete Procedure

After you type the following routine into LOGO's work-space, you can use it to remove an element from a list:

```
TO REMOVE :OBJECT :ELEMENT
     IF MEMBER? :ELEMENT :OBJECT
     [MAKE "PLACE WHERE
     MAKE "COUNT COUNT :OBJECT
       SELECT [
         :PLACE=1 [MAKE "OBJECT
          BUTFIRST :OBJECT]
         :PLACE=COUNT :OBJECT [MAKE
          "OBJECT BUTLAST :OBJECT]
         :PLACE>1 [MAKE "OBJECT SE
          (PIECE 1 :PLACE-1 :OBJECT)
          (PIECE :PLACE+1 COUNT
          :OBJECT :OBJECT)]]]
     ELSE [PRINT :ELEMENT [IS NOT A
      MEMBER OF]]
     PRINT :OBJECT
END
```

To use this procedure, first create a list variable, using any name you wish. An example is:

```
? MAKE "MYLIST [ONE TWO THREE FOUR
FIVE SIX SEVEN] ENTER
```

Now execute the procedure by typing: **REMOVE :MYLIST "THREE**. REMOVE displays the new list ONE TWO FOUR FIVE SIX SEVEN. The element THREE is removed. When you execute this procedure, the list you input to the procedure (in this example, MYLIST) is transferred to the variable OBJECT. The word you input to the procedure (THREE) is transferred to the variable ELEMENT. The MEMBER? primitive in Line 2 determines whether ELEMENT is a member of the list. If it is, the location of ELEMENT is calculated by WHERE and placed in the variable PLACE.

The SELECT primitive selects an operation to perform, according to the value of PLACE.

- If PLACE is 1, the procedure removes the first element from the list you specified.
- If PLACE equals COUNT, the element you specified is the last one in the list, and the procedure removes the last element.
- If PLACE points to neither the first nor the last element and is not 0, the procedure calls PIECE into action. First, PIECE selects all the items up to the one you want removed. Then it selects all the items after the one you want removed. Last, it joins these 2 lists to form the new list, with the appropriate element deleted.

The last SELECT line is quite confusing but, if you break it into its separate elements, it is not hard to understand:

PIECE 1 :PLACE–1 :OBJECT
Selects all elements, from the beginning of the list to the element immediately before the one specified for deletion.

*. . . . **About Quotes***

You can use several quotation marks in a line you want to display. However, because LOGO expects a word to follow a quotation mark, it only displays every other one.
PRINT
"QUOTES" " " " " " "
results in:

QUOTES " " "

PIECE :PLACE+1 COUNT :OBJECT
Selects all the element's that follow the word you specified for deletion.

MAKE "OBJECT SE
creates a new sentence of the 2 selections that now excludes the word you specified for deletion.

Insert Procedure

The following procedure operates in the same manner as REMOVE except it inserts, rather than extracts, an element from a list. If PLACE is 1, then the procedure inserts a new element at the beginning of the list. If PLACE is greater than COUNT (the last element in the list), then the procedure adds a new element.

If PLACE falls between the beginning and the end of the list, then PIECE selects all the members before the insert position and adds a new element. PIECE then selects all the members following the insert position and appends them.

You can use this routine to insert the new element TEST at Position 3 in the list MYLIST by typing:

```
INSERT :MYLIST "TEST 3 ENTER
```

The routine for INSERT follows:

```
TO INSERT :OBJECT :ELEMENT :PLACE
     SELECT [
          :PLACE=1 [MAKE "OBJECT
           (FPUT :ELEMENT :OBJECT)]
          :PLACE=COUNT :OBJECT [MAKE
           "OBJECT (LPUT :ELEMENT
           :OBJECT)]
```

```
                :PLACE>1 [MAKE "OBJECT
                 SENTENCE (PIECE 1 :PLACE-1
                 :OBJECT) :ELEMENT (PIECE
                 :PLACE COUNT :OBJECT
                 :OBJECT)]]
                PRINT :OBJECT
        END
```

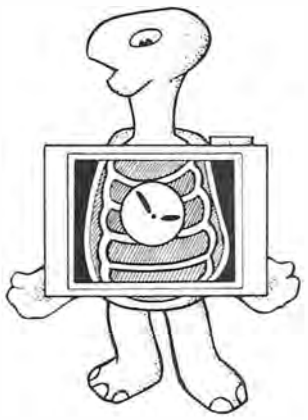
Section 5
Running Away With Lists

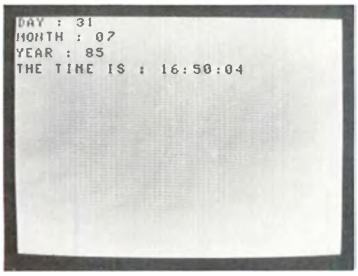
The primitive RUN can be used to execute 1 procedure from within another procedure. For instance, if your workspace contains the procedures ADD, SUB, MULT, and DIV, you can make a list of these names and use RUN to execute any of them. To do so, use list-manipulation primitives to extract the name of the procedure you wish to execute. The following procedure demonstrates this technique:

```
TO MENU
     CLEARTEXT
     PRINT [* * * * * * M E N U * *
      * * * *]
     PRINT
     PRINT [1. ADDITION]
     PRINT [2. SUBTRACTION]
     PRINT [3. MULTIPLICATION]
     PRINT [4. DIVISION]
     PRINT
     PRINT1 [PRESS NUMBER OF
      CHOICE...]
     MAKE "CHOICE RC
     PRINT :CHOICE
     RUN LIST ITEM :CHOICE [ADD SUB
      MULT DIV]
END

TO ADD
     PRINT [THIS COULD BE THE
      ADDITION ROUTINE]
END

TO SUB
     PRINT [THIS COULD BE THE
      SUBTRACTION ROUTINE]
END
```

```
TO MULT
     PRINT [THIS COULD BE THE
      MULTIPLICATION ROUTINE]
END

TO DIV
     PRINT [THIS COULD BE THE
      DIVISION ROUTINE]
END
```

Line 12 in the MENU routine uses ITEM and LIST to separate the operation you chose from all the operations available. The RUN primitive then executes your selection. Using RUN with the name of a procedure is the same as executing a procedure by typing its name. Suppose you select item 3 from the sample menu. Line 12, in effect, then reads: RUN [MULT].





```
DAY : 31
MONTH : 07
YEAR : 85
THE TIME IS : 16:50:04
```

Section 6 The Turtle Swallowed A Clock

D.L. LOGO can take advantage of the built-in OS-9 time and date function. The primitive DATE returns the date and time in the following format: MM/DD/YY hh:mm:ss.

If the present time and date is 12:45:30, May 28, 1985, typing DATE ENTER displays:

```
    05/28/85 12:45:30
```

Using the PIECE primitive, you can extract any element of the time and create a clock for your Turtle. The following procedure does this:

```
    TO TIME
         CLEARTEXT
         LABEL "START
         SETCURSOR 0 0
         PRINT [DAY:] PIECE 4 5 DATE
         PRINT [MONTH:] PIECE 1 2 DATE
         PRINT [YEAR:] PIECE 7 8 DATE
         PRINT [THE TIME IS:] PIECE 10
          17 DATE
         GO "START
    END
```

The procedure extracts all elements of the date and time and displays them on the text screen. The procedure then uses a GO-LABEL loop to continuously update the time. However, if you do not enter the correct date and time when you initialize D.L. LOGO, the date and time created by this procedure is not correct.

In addition to creating a very expensive watch, the date and time feature serves many other purposes. You can use the time values to create greater "randomness" in the

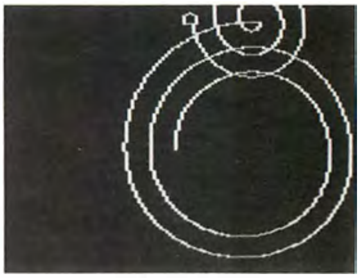
RANDOM primitive. For instance, the following procedure makes a predictable random sequence unlikely:

```
? MAKE "X (RANDOM 10)* (PIECE 16 17
DATE)
```

Because the seconds' value is constantly changing, they add another unpredictable element to the random function.

You can also use the values representing minutes and seconds to time games or quizzes. You can work time values into graphics displays to create graphics time images such as the following:

```
TO TIMELOOP
     REPEAT 1500 [MAKE "T PIECE 16
          17 DATE
         FD :T/20 RT 2]
END
```



. . . . About Parentheses

You use parentheses in command statements to clarify the order of operations. The procedure, as well as the argument, must be enclosed within the parentheses. Following is an example of a statement with, and without parentheses:

```
MAKE "R (RANDOM
   11)+1
MAKE "R RANDOM
   11+1
```

The first statement creates a random number in the range 1-11. The second statement creates a random number in the range 0-11.

Chapter Summary

PRIMITIVE	Abbrev.	Purpose
WORD	–	combines words to create 1 word.
LIST	–	creates lists from other lists, words, or sentences.
WORD?	–	determines whether an object is a word.
LIST?	–	determines whether an object is a list.
FIRST	–	extracts the first element of an object.
LAST	–	extracts the last element of an object.
BUTFIRST	–	extracts all but the first element of an object.
BUTLAST	–	extracts all but the last element of an object.
FPUT	–	inserts an element at the front of an object.
LPUT	–	inserts an element at the end of an object.
ITEM	–	extracts a specified element from a specified location in a word or list.

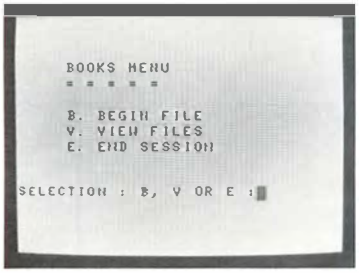
PRIMITIVE	Abbrev.	Purpose
PIECE	–	extracts a specified number of elements from a specified location in a word or list.
COUNT	–	returns the number of members in a word or list.
MEMBER?	–	determines whether a specified element is a member (is included) in an object.
WHERE	–	locates the position of an element in a word or list.
EMPTY?	–	determines if an object has 0 members.
RUN	–	executes a procedure name contained in a list.
DATE	–	returns the current date and time.

Turtle Facts

- A *word* is a string (group) of 1 or more characters.
- A *list* is a string of 1 or more words.
- A *sentence* combines words and lists into a single list.
- A word, list, or sentence can have as few as 0 elements or as many elements as your computer's memory allows.

. . . . About a Library

Because you can use the procedures you create in exactly the same manner as a LOGO primitive, you can begin to establish your own LOGO library. This manual has many procedures that you can use over and over in various programs. As you study and write many of your own procedures, you might want to format a separate diskette for your own library. Every time you create or come across a useful procedure, give it an appropriate name and save it on your Library diskette. If your diskette becomes full, begin a new one. When you need a specific program, you can bypass much of the work by using the APPEND primitive to link library procedures.



```
  BOOKS MENU
  = = = = =

  B. BEGIN FILE
  V. VIEW FILES
  E. END SESSION

SELECTION : B, V OR E :
```

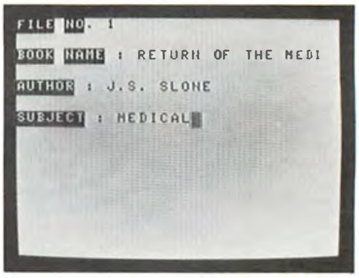
- Quotation marks or the primitive QUOTE indicate a word. A space indicates the end of a word.
- An empty word or list is a word or list that has 0 members.
- Numbers are words.
- Square brackets ([ ]) indicate lists.
- You can include lists inside of lists.

Suggested Project

Use the word and list manipulating techniques you have learned to create a filing program for a book library. Have the program provide 3 fields of data, the book name, the author, and the subject. Do not worry about saving your files on diskette at this stage.

Suggested Solution

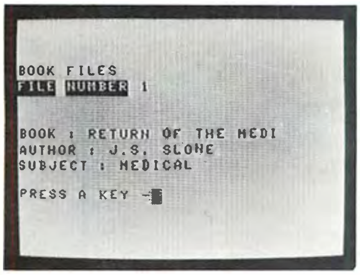
```
TO BOOKS
    MAKE "NAME [BOOK FILES]
    MAKE "CT 0
    MENU
END

TO MENU
    CLEARTEXT
    SETCURSOR 3 5
    PRINT [BOOKS MENU]
    SETCURSOR 4 5
    PRINT [======]
    SETCURSOR 6 5
    PRINT [B. BEGIN FILE]
    SETCURSOR 7 5
    PRINT [V. VIEW FILES]
    SETCURSOR 8 5
    PRINT [E. END SESSION]
    SETCURSOR 9 5
    P 2
    PRINT1 [SELECTION: B V OR E?:]
    CLEARINPUT
    MAKE "CHOICE RC
    SELECT [
        :CHOICE = "B [INPUT]
        :CHOICE = "V [VIEW]
        :CHOICE = "E [QUIT]]
END

TO INPUT
    CLEARTEXT
    PRINT [file no. ] :CT+1
    P 1
    PRINT [book name: ]
    P 1
    PRINT [author: ]
    P 1
    PRINT [subject: ]
```



```
FILE NO. 1

BOOK NAME : RETURN OF THE MEDI

AUTHOR : J.S. SLONE

SUBJECT : MEDICAL
```

. . . . *About Books*

When this program is typed, execute it by typing from the immediate mode, BOOKS ENTER. *A menu is displayed. Choose the B option to begin a new file. When files are created, choose the V option to view them. The E option causes the program to finish. When you begin a file, BOOKS prompts you to type a book name and the author's name and to define the subject. Press* ENTER *after each of the items you type. To quit making entries, type END instead of a bookname, and the menu reappears.*



```
BOOK FILES
FILE NUMBER 1

BOOK : RETURN OF THE MEDI
AUTHOR : J.S. SLONE
SUBJECT : MEDICAL

PRESS A KEY -
```

```
    SETCURSOR 2 12
    MAKE "BOOKNAME RQ
    IF :BOOKNAME = [END] [MENU]
    SETCURSOR 4 9
    MAKE "AUTHOR RQ
    SETCURSOR 6 10
    MAKE "SUBJECT RQ
    MAKE "CT :CT+1
    COMBINE
END

TO COMBINE
    MAKE (WORD "FILE :CT) LIST :BOOKNAME
     :AUTHOR :SUBJECT
    INPUT
END

TO VIEW
    MAKE "C 0
    LABEL "NEXT
    MAKE "C :C+1
    IF :C>:CT [ENDVIEW]
    MAKE "SHOW (WORD "FILE :C)
    CLEARTEXT
    PRINT :NAME
    P3
    PRINT [file number] :C
    P1
    PRINT [BOOK:] FIRST THING :SHOW
    PRINT [AUTHOR:] ITEM 2 THING :SHOW
    PRINT [SUBJECT:] LAST THING :SHOW
    P1
    PRINT1 [PRESS A KEY - ]
    MAKE "NUL RC
    GO "NEXT
END
```

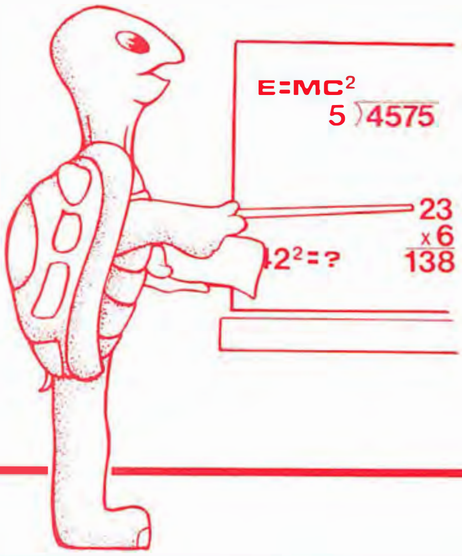
```
TO ENDVIEW
    SETCURSOR 12 0
    PRINT1 [END OF FILES...]
    MAKE "NUL RC
    MENU
END

TO QUIT
    CLEARTEXT
    SETCURSOR 7 8
    PRINT [SESSION OVER...]
    TOPLEVEL
END

TO P :T
    REPEAT :T [PRINT]
END
```

*. . . . **About Objects***

A Logo procedure can require a specified number and kind of inputs to perform its task. These inputs, called objects *can be one or more words, lists, or numbers.*

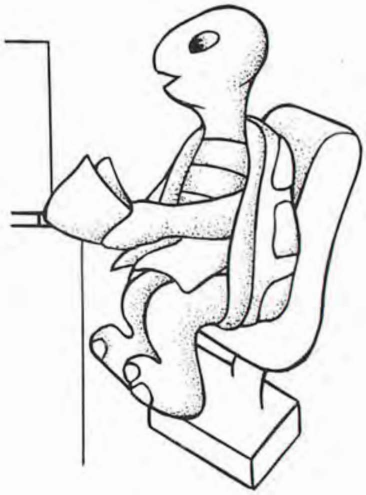
9 A CALCULATING TURTLE

Working with Numbers



Section 1	The Numbers Game: a look at what Turtle can do with numbers.
Section 2	Turtle Figures: how the Turtle does its arithmetic.
Section 3	Comparing Figures and Facts: comparing data.
Section 4	Calculating with Precision: from integers to 100-place calculations.
Section 5	Making Rearrangements: random numbers, shuffling, and sorting.

Section 1 The Numbers Game



You have seen numerous examples of how the Turtle can handle calculations. D.L. LOGO is equipped to do much more. Other calculating features let you:

- Set the precision of operations to a maximum of 100 places
- Add, subtract, divide, multiply, and exponentiate
- Test for equal, greater than, less than as well as create random numbers and shuffle elements in lists
- Find sums, products, quotients, and remainders
- Calculate logarithms, exponents, sines, cosines, tangents, arctangents, square roots, and absolute values
- Round off, create integers, and remove integer portions
- Use logical AND, OR, NOT, and ELSE functions on true/false statements
- Determine the ASCII code for a character
- Determine the character for a given ASCII code
- Determine whether a word is a number
- Accomplish nearly any logical, mathematic, or comparative operation by using D.L. LOGO's *built in* arithmetic operations

This chapter contains information that is essential to a comprehensive understanding of D.L. LOGO. Take your time to study the explanations, try the examples, and experiment with the various concepts. The mathematic, comparative, and logical concepts you learn are needed in most procedures and programs you write. Understanding these concepts makes programming in LOGO easier.

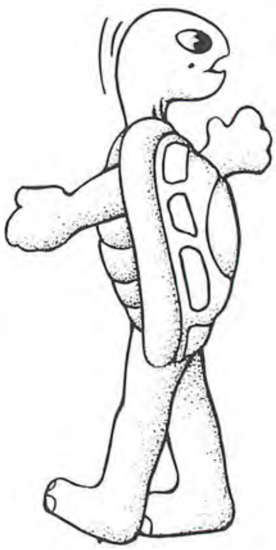
At the same time, remember that you can do a great deal of programming and experimentation with simple mathematic and logical concepts. If some of the material seems too difficult, pass over it now. As you gain more experience, you can come back and better understand the more difficult concepts.

Section 2 Turtle Figures

The following table lists D.L. LOGO's arithmetic and comparative operators, along with their functions and the objects they accept.

All operators require 2 objects, and they all return 1 object, the result of the operation (for instance 2+3=6). The arithmetic operators return a number, and the comparison operators return TRUE or FALSE.

Symbol and Operation		Example	Object accepted
Arithmetic Operators			
+	Addition	2+2	Numbers
–	Subtraction	2–2	Numbers
*	Multiplication	2*2	Numbers
/	Division	2/2	Numbers
↑	Raise to a power *	2^2	Numbers



Symbol and Operation		Example	Object accepted
Comparative Operators			
=	Test for equal	:A=65	Any Objects**
>	Test for greater than	BILL>SAM	Any Words**
<	Test for less than	A<:B	Any Words**
>=	Test greater than or equal	:A<=0	Any Words**
<=	Test for less than or equal	:A<=:B	Any Words**

* Pressing CTRL and 7 simultaneously creates ↑.
** D.L. LOGO compares numbers numerically; it compares objects that are not numbers alphabetically.

A Turtle Calculator — Standard Equipment

Because your Turtle has a built-in calculator, stumping it with a tough math question is nearly impossible. For instance, if you wish to draw a 12-sided figure on the screen but don't know the necessary angle to accomplish this, enter the immediate mode and type:

```
? 360/12 ENTER
```

The screen displays *30*.

D.L. LOGO in the immediate mode can handle very simple or very complicated computations. This question is quite simple, but feel free to try to stump your Turtle with a really tough query. Once you know how to ask your Turtle to calculate square roots, you might ask for the square root of 4567112349.41299 divided by 44. Your pocket calculator can't handle that calculation very easily, but D.L. LOGO can. This chapter tells you how you can ask your Turtle tough questions and get the right answers.

By the way, Turtle says the answer to the preceding question is 1535.909. Continue reading for more ways to get this kind of information from the Turtle.

Using Proper Syntax

Although D.L. LOGO's basic arithmetic operations are clear, it is important to understand how syntax can affect an expression. The rules are logical and easy to follow.

To use an arithmetic operator on 2 numbers, separate the numbers with the operator. For instance, in the immediate mode you can type:

```
? 12+12 ENTER
24
```

Or you can type:

```
? 12 + 12 ENTER
24
```

You can include or omit spaces. They do not affect the operation.

Operation	Result
3 +4	7
4 * 3	12
4/ 2	2
4↑2	16

Because the minus sign (–) can indicate either a negative number or a subtractive operator, special rules govern this symbol, and spaces do affect its operation. D.L. LOGO assumes that the minus sign is a subtractive operator unless a number immediately follows it, but does *not* immediately precede it:

. . . . ***About numbers***

*LOGO handles numbers in lists and words in the same manner as it handles alphabetic characters. You can use numbers in variable names and as text. However, when you include arithmetic or logic operators (+ – = * / < > ↑) in a word, they cause LOGO to perform the indicated arithmetic operation.*

Arithmetic operators in lists do not *cause the indicated operation to be performed. LOGO treats arithmetic symbols in lists in the same manner as other characters except they are displayed with a leading space. For example, PRINT [5*5 is 10] produces 5 *5 is 10.*

. . . . About Operations

The accompanying references show examples using numbers, but LOGO handles variables in exactly the same manner. For example, the operation :A + :B is just as valid as 1 + 2, as long as both A and B are defined as numeric variables.

Operation	Result
5−2	3
5 −2	5 −2
5 − 2	3
5 −2	3
5+ −2	3
(4+1) −2	3
(4+1) −2	5 −2

True or False

D.L. LOGO's comparative operations function on both numbers and alphabetic characters. Numeric operations compare numeric values. The following table shows the result of greater-than, less-than, and equal comparisons:

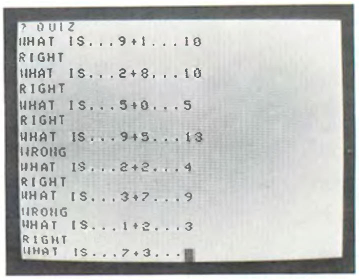
Value 1	Operator	Value 2	Result
1	>	2	FALSE
1	<	2	TRUE
5	<>	Ø	TRUE
5	=	5	TRUE
5	>=	1	TRUE
5	<=	1	FALSE
5	<=	5	TRUE

To demonstrate the construction of this type of logic, read the preceding lines as:

Line 1:	1 is greater-than 2? FALSE
Line 2:	1 is less-than 2? TRUE
Line 3:	5 is greater-than or less-than Ø? TRUE
Line 4:	5 is equal to 5? TRUE
Line 5:	5 is greater-than or equal to 1? TRUE
Line 6:	5 is less-than or equal to 1? FALSE
Line 7:	5 is less-than or equal to 5? TRUE

A practical application of such logic is an addition quiz:

```
TO QUIZ
     REPEAT 10 [
          MAKE "N1 RANDOM 10
          MAKE "N2 RANDOM 10
          MAKE "A :N1 + :N2
          PRINT1 [WHAT IS ...]:N1[+]
           :N2[...]
          MAKE "A1 FIRST RQ
          SELECT [
            :A = :A1 [PRINT[RIGHT]]
            :A <> :A1 [PRINT [
             WRONG]]]]
     PRINT [THAT'S ALL]
END
```



```
? QUIZ
WHAT IS...9+1...10
RIGHT
WHAT IS...2+8...10
RIGHT
WHAT IS...5+0...5
RIGHT
WHAT IS...9+5...13
WRONG
WHAT IS...2+2...4
RIGHT
WHAT IS...3+7...9
WRONG
WHAT IS...1+2...3
RIGHT
WHAT IS...7+3...
```

The SELECT primitive compares the correct answer, A, to your answer, A1. If they are equal (=), the screen displays the word "RIGHT." If A is less than or greater than A1, the screen displays the word "WRONG." The above procedure asks 10 questions, and then ends by displaying "THAT'S ALL."

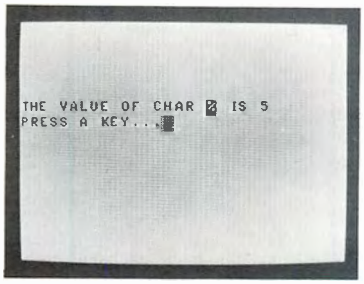
ASCII and the Alphabet

When comparing characters and words, D.L. LOGO compares ASCII values. For instance, the ASCII code for *A* is 65, and the ASCII code for *B* is 66. To compare 2 characters, precede each character with a quotation mark, such as **PRINT "F < "Z**. To compare variables, type the variable names preceded by dots: **PRINT :MONEY > :LOVE**. Variables must be defined or such a comparison will cause an UNDEFINED SYMBOL error. For instance:

```
? MAKE "MONEY "$1,000,000.00 ENTER
? MAKE "LOVE "JANE ENTER
? PRINT :LOVE > :MONEY ENTER
TRUE
```

. . . . *About ASCII*

ASCII is the abbreviation for American Standard Code for Information Interchange. The code gives a standard numeric value to computer generated characters. This allows for the exchange of characters between computers and other devices, such as modems, printers and plotters. D. L. LOGO uses this code in comparing the values of characters.



```
THE VALUE OF CHAR ▨ IS 5
PRESS A KEY...
```

. . . . About Characters 10 and 13

Character 10 causes a linefeed in D.L. LOGO. Character 13 causes a linefeed and carriage return. When the program attempts to display these characters, they cause the display to drop to Line 6, rather than the normal Line 5 display.

Following is a chart of sample comparisons:

Character 1	Operation	Character 2	Result
A	>	B	FALSE
A	<	B	TRUE
A	=	A	TRUE
?	<	.	FALSE
B	<	A	FALSE
C	<>	B	TRUE
C	>=	B	TRUE

When comparing words that have more than 1 character, LOGO looks at the corresponding characters of each word until it finds a difference. For instance, to compare PEACH to PEACHES, LOGO sequentially looks at each letter in each word until it finds an unequal pair. In this example, PEACH is of lesser value than PEACHES because it has fewer characters.

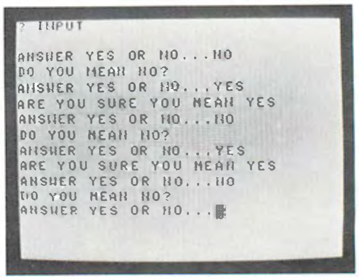
In like comparisons, BREAD is of lesser value than BUTTER, and HORSE is of greater value than COW. When LOGO compares alphabetic characters, it gives the least value to the character that occurs first in the alphabet. To see all the character values in D.L. LOGO, enter and execute this program:

```
TO CHARACTER
     CLEARTEXT
     FOR "T 1 255 1 [SETCURSOR 5 0
          PRINT [THE VALUE OF
            CHARACTER] CHAR :T [IS]
            :T
          PRINT1 [PRESS A KEY...]
          MAKE "NUL RC]
END
```

A keyboard input provides a practical application of a word comparison routine. The following procedure produces a menu of certain functions. LOGO determines what function you select by comparing the character you select to a list.

```
TO INPUT
        PRINT
        PRINT1 [ANSWER YES OR NO...]
        MAKE "I FIRST RQ
        SELECT [
                :I="YES [PRINT [ARE YOU
                  SURE YOU MEAN YES?]
                  INPUT]
                :I="NO [PRINT [DO YOU MEAN
                  NO? [INPUT]
                :I<>" [PRINT [YOU MUST
                  TYPE YES OR NO!] INPUT]]
END
```

The RQ primitive in Line 3 accepts keyboard entries until you press ENTER. LOGO compares your input, contained in the variable I, to the words "YES and "NO and to an empty word.



```
? INPUT

ANSWER YES OR NO...NO
DO YOU MEAN NO?
ANSWER YES OR NO...YES
ARE YOU SURE YOU MEAN YES
ANSWER YES OR NO...NO
DO YOU MEAN NO?
ANSWER YES OR NO...YES
ARE YOU SURE YOU MEAN YES
ANSWER YES OR NO...NO
DO YOU MEAN NO?
ANSWER YES OR NO...█
```

. . . . *About RQ*

The primitive RQ (REQUEST) returns input in the form of a list, rather than a word. To compare an RQ input with a word, you must either convert the RQ input to a word or convert your word to a list. In the accompanying INPUT procedure, the FIRST primitive is used to convert the input into a word. FIRST extracts the first element (a word) from the RQ list.

More Arithmetic

D.L. LOGO handles many other functions in addition to basic math and comparisons. The following reference shows these functions, their syntax, special notes or suggestions, and examples of their use.

SUM

Purpose: adds a series of numbers.

Notes:

- You can use only numbers with SUM.
- SUM adds any number of inputs.
- Requesting the SUM of a list of numbers gives the same result as including a plus sign (+) between each number.

Example: SUM 1 2 3 4 5
Result: 15

PRODUCT

Purpose: multiplies a series of numbers.

Notes:

- You can use only numbers with PRODUCT.
- PRODUCT multiplies any number of inputs.
- Requesting the PRODUCT of a series of numbers is the same as placing a multiplication sign between the numbers.

Example: PRODUCT 1 2 3 4 5
Result: 120

QUOTIENT

Purpose: calculates the whole number result of dividing one number by another.

Notes:

- QUOTIENT requires 2 inputs, the dividend and the divisor.
- Each input is rounded to the nearest whole number before the division takes place. (Numbers with a fractional portion less than 0.5 are rounded down, and numbers with a fractional portion of 0.5 or greater are rounded up.)
- If the result has a decimal portion, it is rounded to the nearest whole number.

Example: QUOTIENT 6.3 2.1
Result: 3

To understand this operation, read it as: the rounded value of 6.3 (6) divided by the rounded value of 2.1 (2) equals 3. Three is a whole number and does not need to be rounded.

Example: QUOTIENT 11.3 3
Result: 4

To understand this operation, read it as: the rounded value of 11.3 (11) divided by 3 equals the rounded value of 3.77 (4).

REMAINDER

Purpose: determines the whole number remainder of one number divided by another.

Notes:

- REMAINDER accepts 2 inputs (the dividend and the divisor).
- The function rounds both inputs to whole numbers before it performs the division. (Numbers with a fractional portion less-than 0.5 are rounded down and numbers with a fractional portion of 0.5 or greater are rounded up.)

Example: REMAINDER 28.6 10
Result: 9

To understand this operation, read it as: the rounded value of 28.6 (29) divided by 10 equals 2 with a remainder of 9.

Example: REMAINDER 44 6.2
Result: 2

To understand this operation, read it as: 44 divided by the rounded value of 6.2 (6) equals 7 with a remainder of 2.

ROUND

Purpose: rounds a mixed number (a number containing a fractional part) to the nearest whole number.

Notes:

- ROUND accepts 1 input, the number you want to round.
- The function rounds down numbers with a fractional portion less than Ø.5 and rounds up numbers with a fractional portion of Ø.5 or greater.

Example: ROUND 5.4
Result: 5 (the nearest whole number)

Example: ROUND 5.5
Result: 6 (because fractional values of Ø.5 or greater are rounded up)

INTEGER

Purpose: reduces a mixed number (a number containing a fractional part), to the next whole number.

Notes:

- INTEGER accepts 1 input only, the number from which the integer portion is to be extracted.

Example: INTEGER 3.123
Result: 3 (the fractional portion is removed)

Example: INTEGER -22.7
Result: – 23 (INTEGER reduces to the next whole number.

FIXED

Purpose: removes the fractional portion of a mixed number.

Notes:

- FIXED accepts 1 input only, the number to be rounded.

Example: FIXED 2.999
Result: 2

Example: FIXED −22.7
Result: −22

FRACTION

Purpose: removes the integer portion of a number.

Notes:

- FRACTION accepts 1 input only, the number from which the function retains the fraction portion.

Example: FRACTION 3.45
Result: 0.45 (the integer portion of the number is removed)

LOG

Purpose: computes the natural log of a number.

Note:

- LOG accepts 1 input only, the number from which the function derives the log.

Example: LOG 2.3
Result: 0.84

Example: LOG 44/12
Result: 1.3

EXP

Purpose: computes *e* raised to the power of a given number.

Notes:

- *e* is 2.71, the base of the LOG function.

- EXP accepts 1 input only, the number that serves as the exponent of *e*.

Example: EXP 1
Result: 2.71

ABS

Purpose: returns the absolute value of a given number.

Notes:

- Absolute value is the the value of a number or variable without regard to its sign (plus or minus).
- ABS accepts 1 input only, the number for which the function calculates the absolute value.

Example: ABS 2.1
Result: 2.1

Example: ABS −2.1
Result: 2.1

COS

Purpose: calculates the cosine of a specified number.

Note:

- COS accepts 1 input only, the angle (in degrees) from which the function calculates the cosine.

Example: COS 6Ø
Result: Ø.49

SIN

Purpose: calculates the sine of a specified number.

Note:

- SIN accepts 1 input only, the angle (in degrees) from which the function calculates the sine.

Example: SIN 60
Result: 0.86

TAN

Purpose: calculates the tangent of a given number.

Note:

- TAN accepts 1 input only, the angle (in degrees) from which the function calculates the tangent.

Example: TAN 44
Result: .97

ATAN

Purpose: calculates the arctangent of a specified ratio.

Note:

- ATAN accepts 2 inputs, the X component and the Y component, and returns the arctangent of X/Y.

Example: ATAN 0.5 1
Result: 26.56

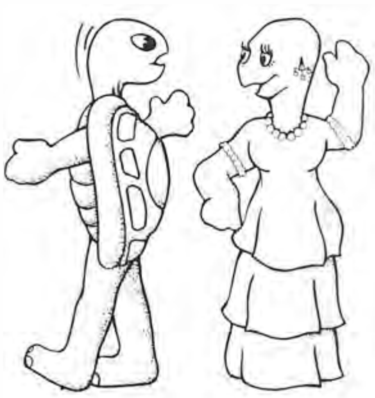
SQRT

Purpose: calculates the square root of a specified number.

Note:

- SQRT accepts 1 input only, the number from which the function calculates the square root.

Example: SQRT 2
Result: 1.41



Section 3 Comparing Figures and Facts

In addition to arithmetic and trigonometric operations, D.L. LOGO also performs Boolean logic comparisons. Following is a list of the functions that it can perform.

ALLOF

Purpose: performs a logical AND operation on a series of true/false operations or statements.

Notes:

- ALLOF accepts any number of true/false operations or words.
- If *all* operations or words are "TRUE," LOGO returns a result of "TRUE".
- If 1 or more of the specified operations or words are "FALSE," LOGO returns a result of "FALSE".

Example: SHOW (ALLOF "TRUE "TRUE "TRUE "TRUE)
Result: TRUE

Example: SHOW (ALLOF "TRUE "TRUE "FALSE "TRUE)
Result: FALSE

Example: SHOW (ALLOF MEMBER? "Y "YES MEMBER? "N "NO MEMBER? "B "MAYBE)
Result: TRUE

Example: SHOW (ALLOF MEMBER? "Y "YES MEMBER? "L "NO MEMBER? "B "MAYBE)
Result: FALSE

ANYOF

Purpose: performs a logical OR on a series of true/false operations or words.

Notes:

- ANYOF accepts any number of operations or words.
- If *any* of the specified operations or words are "TRUE," LOGO returns a result of "TRUE".
- If *all* of the specified operations or words are "FALSE," LOGO returns a result of "FALSE".

Example: SHOW (ANYOF "TRUE "FALSE "TRUE)
Result: TRUE

Example: SHOW (ANYOF "FALSE "FALSE "FALSE)
Result: FALSE

Example: SHOW (ANYOF MEMBER? "Y "YES MEMBER?
"L "NO MEMBER? "G "MAYBE)
Result: TRUE

Example: SHOW (ANYOF MEMBER? "B "YES MEMBER?
"L "NO MEMBER? "G "MAYBE)
Result: FALSE

NOT

Purpose: performs a logical complement of an operation or a word.

Notes:

- NOT accepts 1 input, the operation or word the function complements.
- If the operation or word is "TRUE," LOGO returns a result of "FALSE".
- If the operation or word is "FALSE," LOGO returns a result of "TRUE".

Example: SHOW NOT "TRUE
Result: FALSE

Example: SHOW NOT (MEMBER? "Y "YES)
Result: FALSE

Example: SHOW NOT (MEMBER? "N "YES)
Result: TRUE

Data Comparisons

Use the following LOGO primitives in data comparisons, generations, and manipulations.

ASCII

Purpose: returns the ASCII code for a specified character.

Notes:

- ASCII accepts 1 input, the character for which to find the equivalent in ASCII code.
- If a word or series of characters are the input, the function returns the ASCII code for the first character.

Example: ASCII "A
Result: 65

Example: ASCII "APPLE
Result: 65

Example: ASCII "T
Result: 84

CHAR

Purpose: returns the character of the specified ASCII code.

Note:

- CHAR accepts 1 input, the ASCII code.

Example: CHAR 65
Result: A

Example: CHAR 84
Result: T

NUMBER?

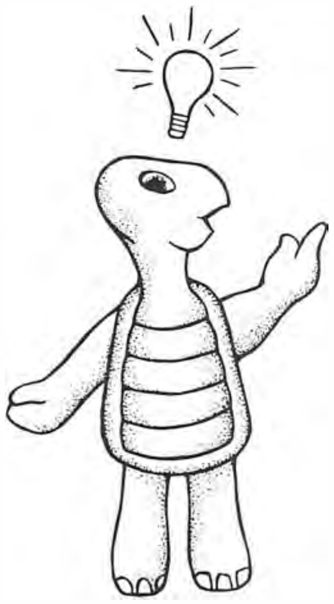
Purpose: determines whether a word is a number.

Notes:

- NUMBER? accepts 1 word as an input.
- If the word is a number, the function returns "TRUE". If the word is not a number, the function returns "FALSE".

Example: NUMBER? 1234
Result: TRUE

Example: NUMBER? "H34
Result: FALSE



Section 4
Calculating with Precision

D.L. LOGO provides mathematical precision to a maximum of 100 places. This means your results can have as many as 100 decimal places and as few as 0. A number with 0 precision is an integer.

To establish the precision of calculations, use the SETPRECISION primitive. For example, to establish a precision of 10 places, type **SETPRECISION 10** ENTER. The square root of 11 is a prime candidate for varying precision. Following are some examples of the calculation of the square root of 11, using precisions of 0, 1, 3, 50, and 100 places:

Square root of 11	
Precision	**Result**
0 place	3
1 place	3.3
3 place	3.316
50 place	3.31662479035539984911493273667068668392708854558935
100 place	3.3166247903553998491149327366706866839270885455893535970586821461164846426090438467088439912829065 09

When you first load D.L. LOGO, the precision is set at 2 decimal places. You can check the current precision at any time with the PRECISION primitive:

```
? PRECISION ENTER
2
```

Section 5
Making Arrangements

D.L. LOGO has several operations that simplify the arranging of data. You can use these operations to produce random lists or to extract random items from a list.

Making Things Random

Computer-generated random numbers are not really random unless an unpredictable element enters into the randomizing operation. For instance, immediately after you load D.L. LOGO, producing 10 random numbers gives this result: 1 2 6 1 8 8 9 1 2 8. You can test this after loading LOGO by typing:

```
? REPEAT 10 [PRINT RANDOM 10] ENTER
```

Because of this predictability, D.L. LOGO has a second random primitive (RANDOMIZE) that introduces an alternate random element to the sequence. To repeat a random sequence, use the RERANDOM primitive.

RANDOM

Purpose: generates a random number.

Notes:

- RANDOM accepts 1 input, the upper limit of the random range.
- An input of 5 generates a random number in the range 0–4.
- The allowable range is −32767 to 32767.

Example: RANDOM 10
Possible result: 6

Example: RANDOM 100
Possible result: 87

RANDOMIZE

Purpose: creates an unpredictable random sequence.

Notes:

- When you first start LOGO and use the RANDOM procedure, the sequence of generated random numbers are predictable.
- Using RANDOMIZE generates a random seed to alter this sequence.
- The procedure requires no input and provides no output.

Example: RANDOMIZE
Result: The operation disrupts the normal sequence of random numbers

RERANDOM

Purpose: resets the initial order of random number generation.

Notes:

- Use RERANDOM if you wish to repeat D.L. LOGO's initial sequence of random numbers.
- RERANDOM requires no argument or input values and provides no output; you type only the primitive name.

```
Example: REPEAT 10 [PRINT RANDOM 10]
         1 2 6 1 8 8 9 1 2 8
         ? REPEAT 10 [PRINT RANDOM 10]
         5 0 9 5 2 2 3 7 1 2
         ? RERANDOM
         ? REPEAT 10 [PRINT RANDOM 10] ENTER
         1 2 6 1 8 8 9 1 2 8
```

Shuffling

Although the SHUFFLE primitive is not really an arithmetic operation, it manipulates both numbers and data.

The following procedure demonstrates how SHUFFLE works:

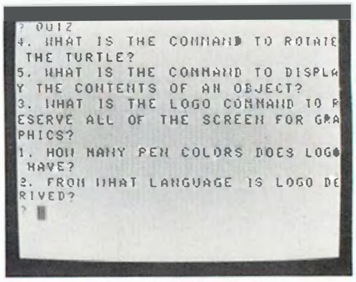
```
? SHUFFLE [1 2 3 4 5 6 7 8 9] ENTER
[3 6 5 1 9 7 8 2 4]
```

You can use the same procedure to shuffle data. For instance, if you wish to create a quiz that randomly selects questions from a list but doesn't repeat any of the questions, use SHUFFLE. The following procedure shows how it works:

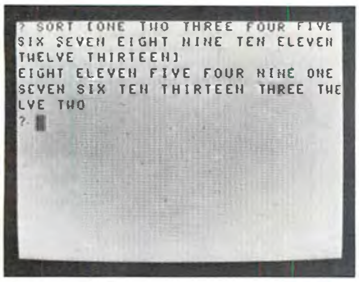
```
TO QUIZ
     MAKE "QUESTIONS LIST [1. HOW
       MANY PEN COLORS DOES LOGO
       HAVE?] [2. FROM WHAT LANGUAGE
       IS LOGO DERIVED?][3. WHAT IS
       THE LOGO COMMAND TO RESERVE
       ALL OF THE SCREEN FOR
       GRAPHICS?][4. WHAT IS THE
       COMMAND TO ROTATE THE
       TURTLE?][5. WHAT IS THE
       COMMAND TO DISPLAY THE
       CONTENTS OF AN OBJECT?]
     MAKE "ASK SHUFFLE :QUESTIONS
     FOR "T 1 5 1 [PRINT ITEM :T
       :ASK]
END
```

From the immediate mode, type **QUIZ** ENTER. The result might look like this:

```
4. WHAT IS THE COMMAND TO ROTATE THE
   TURTLE?
```



```
? QUIZ
4. WHAT IS THE COMMAND TO ROTATE
 THE TURTLE?
5. WHAT IS THE COMMAND TO DISPLA
Y THE CONTENTS OF AN OBJECT?
3. WHAT IS THE LOGO COMMAND TO R
ESERVE ALL OF THE SCREEN FOR GRA
PHICS?
1. HOW MANY PEN COLORS DOES LOGO
 HAVE?
2. FROM WHAT LANGUAGE IS LOGO DE
RIVED?
? 
```



```
? SORT [ONE TWO THREE FOUR FIVE
SIX SEVEN EIGHT NINE TEN ELEVEN
TWELVE THIRTEEN]
EIGHT ELEVEN FIVE FOUR NINE ONE
SEVEN SIX TEN THIRTEEN THREE TWE
LVE TWO
?
```

```
3. WHAT IS THE LOGO COMMAND TO
   RESERVE ALL OF THE SCREEN FOR
   GRAPHICS?
1. HOW MANY PEN COLORS DOES LOGO
   HAVE?
5. WHAT IS THE COMMAND TO DISPLAY
   THE CONTENTS OF AN OBJECT?
2. FROM WHAT LANGUAGE IS LOGO
   DERIVED?
```

Sorting It Out

Below is a program that sorts a list alphabetically. This simple, slow operation repeatedly extracts the smallest element from a list and adds it to a new list until the old list is empty. The following program requires that you input a list of objects before you execute it, such as **SORT [PEACHES PEARS APPLES GRAPES ORANGES PLUMS FIGS]** ENTER.

```
TO SORT :L
          MAKE "M []
          MAKE "CT COUNT :L
          FOR "X 1 :CT 1 [
            MAKE "P 1
            SMALL
            MAKE "M LPUT :WORD :M
            DELETE
            MAKE "CT COUNT :L ]
          PRINT :M
          END

TO SMALL
     MAKE "WORD ITEM 1 :L
     FOR "T 2 :CT 1 [
          IF ITEM :T :L < :WORD
            [MAKE "WORD ITEM :T
              :L MAKE "P :T]]
END
```

```
TO DELETE
     SELECT [
          :P = COUNT :L [MAKE "L
            BUTLAST :L]
          :P = 1 [MAKE "L BUTFIRST
            :L]
          :P > 1 [MAKE "L SE PIECE 1
            :P-1 :L
            PIECE :P+1 :CT :L]
     ]
END
```

The preceding sort program makes use of primitives and concepts from the last 2 chapters. There is also a sort program on your D.L. LOGO diskette that is shorter and quicker, and that uses more advanced concepts.

Chapter Summary

PRIMITIVE	Abbrev.	Purpose
SUM	–	Adds a series of numbers.
PRODUCT	–	Multiplies a series of numbers.
QUOTIENT	–	Divides a number by another.
REMAINDER	–	Determines the whole number remainder of a quotient.
ROUND	–	Rounds a number that contains a fractional portion to the nearest whole number.
INTEGER	–	Reduces a number to the nearest whole number.
FIXED	–	Removes the fractional portion of a fractional number.
FRACTION	–	Removes the integer portion of a number.
LOG	–	Computes the natural log of a number.

PRIMITIVE	Abbrev.	Purpose
EXP	–	Raises a given number to the power of its exponent.
ABS	–	Returns the absolute value of a given number.
COS	–	Calculates the cosine of a specified number.
SIN	–	Calculates the sine of a specified number.
TAN	–	Calculates the tangent of a specified number.
ATAN	–	Calculates the arctangent of a specified ratio.
SQRT	–	Calculates the square root of a specified number.
ALLOF	–	Performs a logical AND operation on a series of true/false operations or statements.
ANYOF	–	Performs a logical OR operation on a series of true/false operations or words.
NOT	–	Performs a logical complement of an operation or word.
ASCII	–	Returns the ASCII code for a specified character.

PRIMITIVE	Abbrev.	Purpose
CHAR	–	Returns the character whose ASCII value equals the specified number.
NUMBER?	–	Determines whether a word is a number.
PRECISION	–	Returns the current precision setting.
SETPRECISION	–	Sets the precision of subsequent operations from 0 to 100 places.
RANDOM	–	Generates a random number.
RANDOMIZE	–	Generates a random initialization reference, or seed, to provide an alternate random sequence.
RERANDOM	–	Resets RANDOM to repeat the initial random order.
SHUFFLE	–	Randomizes a list.

Turtle Facts

- D.L. LOGO uses the following symbols for arithmetic operations:

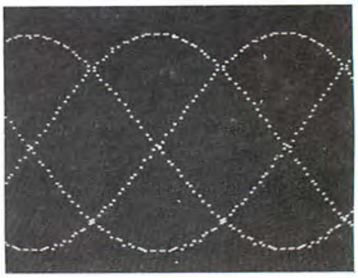
+ Addition	– Subtraction
* Multiplication	/ Division
> Greater-than	< Less-than
= Equal	>= Greater-than or equal
<= Less-than or equal	↑ Exponentiate

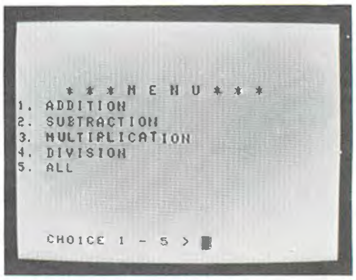
- You create the ↑ by pressing CLEAR and 7 at the same time.
- LOGO compares numbers numerically and other objects alphabetically.
- D.L. LOGO assumes that the minus sign (–) is a subtractive operator unless it is immediately followed by a number and is not immediately preceded by a number.
- Each LOGO character or symbol has an ASCII code.
- LOGO compares nonnumeric objects by their ASCII values.
- D.L. LOGO has a precision range of 0 to 100 places.

Suggested Project

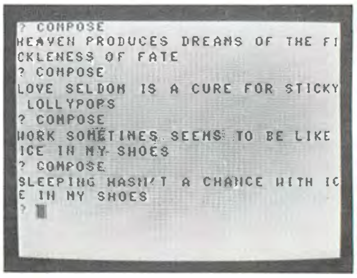
The new concepts in this chapter are very important. Almost every LOGO procedure contains calculations. You can use the following 3 suggestions to see whether you have mastered the basics of D.L. LOGO's comprehensive calculating and comparative operations.

- Write a program to create a sine wave on the graphics screen.





```
      * * * M E N U * * *
1. ADDITION
2. SUBTRACTION
3. MULTIPLICATION
4. DIVISION
5. ALL


   CHOICE  1  -  5 >
```



```
? COMPOSE
HEAVEN PRODUCES DREAMS OF THE FI
CKLENESS OF FATE
? COMPOSE
LOVE SELDOM IS A CURE FOR STICKY
 LOLLYPOPS
? COMPOSE
WORK SOMETIMES SEEMS TO BE LIKE
ICE IN MY SHOES
? COMPOSE
SLEEPING HASN'T A CHANCE WITH IC
E IN MY SHOES
?
```

- Write a quiz that poses problems in addition, subtraction, multiplication, or division, or that randomly selects from all categories.

- Write a program to generate random sentences from lists of subjects, verbs, and objects.

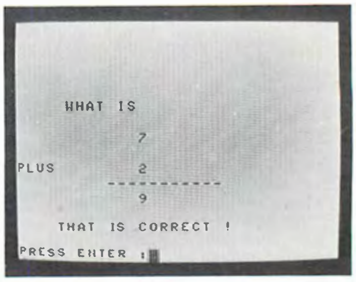
If you have trouble with these projects, the next chapter provides more information and examples using logic, comparison, and calculations. It specifically uses these operations in loops.

Suggested Solution No. 1

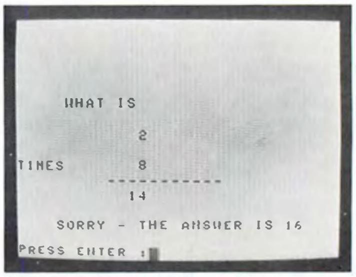
```
TO SINE
     FULLSCREEN
     CS HT WRAP
     MAKE "X 1
     MAKE "B -96
     REPEAT 360 [
          MAKE "A (SIN :X)*75
          DOT :B :A
          MAKE "B :B+2.85
          MAKE "X :X+3
     ]
END
```

Suggested Solution No. 2

```
TO MATH
     CLEARTEXT MENU
     REPEAT 10 [PROBLEM]
     MATH
END

TO MENU
     DO [
     CLEARTEXT
     SETCURSOR 4 5
     PRINT [*** M E N U ***]
     PRINT [1. ADDITION]
     PRINT [2. SUBTRACTION]
     PRINT [3. MULTIPLICATION]
     PRINT [4. DIVISION]
     PRINT [5. ALL]
     SETCURSOR 14 3
     PRINT1 [CHOICE 1 - 5 >\ ]
     MAKE "CH FIRST RQ]
     WHILE (NOT MEMBER? :CH "12345)
END
```



```
     WHAT IS
                7
PLUS            2
           -----------
                9
    THAT IS CORRECT !
PRESS ENTER :
```



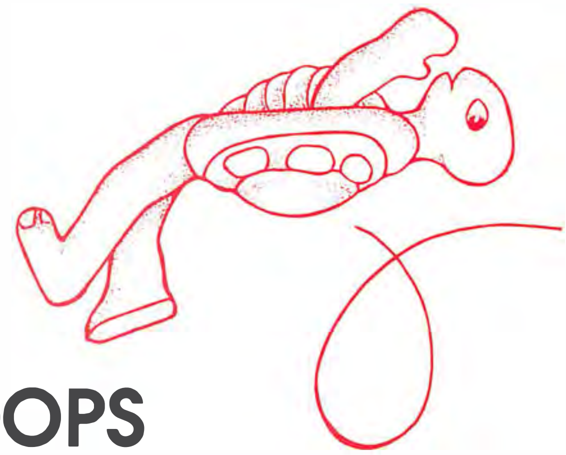
```
     WHAT IS
                2
TIMES           8
           -----------
               14
    SORRY - THE ANSWER IS 16
PRESS ENTER :
```

```
TO PROBLEM
     CLEARTEXT
     SETCURSOR 4 5
     CHOOSE
     DISPLAY
     SELECT [
       :C < 10 [SETCURSOR 11 12]
       :C > 9 [SETCURSOR 11 11]
     MAKE "ANSWER FIRST RQ
     SELECT [
          :ANSWER = :C [SETCURSOR 13 4
            PRINT [THAT IS CORRECT!]]
          :ANSWER <> :C [SETCURSOR 13 4
            PRINT [SORRY - THE ANSWER
            IS] :C]]
     SETCURSOR 15 0
     PRINT1 [PRESS ENTER:]
     CLEARINPUT
     MAKE "Z RC
END

TO ADD
     MAKE "SYMBOL "PLUS
     MAKE "A RANDOM 21
     MAKE "B RANDOM 21
     MAKE "C :A + :B
END

TO SUB
     MAKE "SYMBOL "MINUS
     MAKE "A RANDOM 51
     MAKE "B RANDOM :A+1
     MAKE "C :A - :B
END

TO MULT
     MAKE "SYMBOL "TIMES
     MAKE "A RANDOM 11
     MAKE "B RANDOM 11
     MAKE "C :A * :B
END
```

```
TO DIV
     MAKE "SYMBOL [DIVIDED BY]
     MAKE "B (RANDOM 10)+1
     MAKE "C RANDOM 11
     MAKE "A :B * :C
END

TO OPERATION
     MAKE "R RANDOM (4)+1
     IF :R = 1 [ADD]
     IF :R = 2 [SUB]
     IF :R = 3 [MULT]
     IF :R = 4 [DIV]
END

TO CHOOSE
     IF :CH = 1 [ADD]
     IF :CH = 2 [SUB]
     IF :CH = 3 [MULT]
     IF :CH = 4 [DIV]
     IF :CH = 5 [OPERATION]
     MAKE "CA COUNT :A
     MAKE "CB COUNT :B
END

TO DISPLAY
     SETCURSOR 5 5
     PRINT [WHAT IS]
     SELECT [
          :CA = 2 [SETCURSOR 7 11]
          :CA = 1 [SETCURSOR 7 12]]
     PRINT :A
     SETCURSOR 9 0
     PRINT1 :SYMBOL
     SELECT [
          :CB = 2 [SETCURSOR 9 11]
          :CB = 1 [SETCURSOR 9 12]]
     PRINT :B
     SETCURSOR 10 9
     PRINT [\-\-\-\-\-\-\-\-\-\-]
END
```

Suggested Solution No. 3

```
TO COMPOSE
     SUBJECT
     VERB
     OBJECT
     PRINT :BEGIN :MIDDLE :END
END

TO SUBJECT
     MAKE "BEGIN FIRST (SHUFFLE [[
       HEAVEN] [A WOMAN] [A MAN] WORK
       SLEEPING LOVE])
END

TO VERB
     MAKE "MIDDLE FIRST (SHUFFLE [[MAKES
       ME THINK OF] [SOMETIMES SEEMS TO
       BE LIKE] [MAKES ME HATE] [NEVER
       HELPS WITH] [SELDOM IS A CURE
       FOR] [PRODUCES DREAMS OF] [HASN'T
       A CHANCE WITH] [USED TO BE GOOD
       FOR]])
END

TO OBJECT
     MAKE "END FIRST (SHUFFLE [[THE
       POWER OF LOVE] [ THE FICKLENESS
       OF FATE] [HAPPINESS] [COLD HOT
       DOGS] [A BROKEN HEART] [STICKY
       LOLLYPOPS] LOVE [ICE IN MY
       SHOES]])
END
```

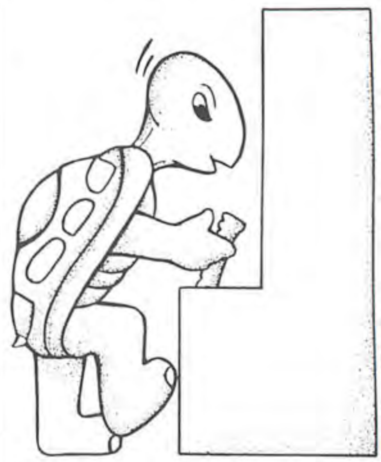


10 LOGIC AND LOOPS

Turtle Logic is Great to Have Around, and Around, and Around

Section 1	Testing, Testing, 1, 2, 3: testing conditions and making selections.
Section 2	A Loop Is a Loop Is a Loop Is a . . . : WHILE, FOR, and GO loops.
Section 3	A Recursive Turtle: the amazing recursive loop.

Section 1
Testing . . . Testing . . . 1 . . . 2 . . . 3



D.L. LOGO provides a number of powerful loop operations. By combining loop functions with comparative logic, you can repeat operations until a condition is true or not true. You can create loops that automatically increase or decrease. You can create your own error-handling routines, test keyboard input, or merge lists, words, or files.

You have already been introduced to some loop and conditional primitives to demonstrate other LOGO functions. Programming even simple procedures can be difficult without loops and conditional primitives. With the material in this chapter and preceding chapters, you can tackle almost any programming job that D.L. LOGO can handle. Section 1 provides more information and examples for using conditional logic. Section 2 describes ways to use conditional logic in loops. Section 3 describes the impressive power of recursive calls.

Testing with IF

The IF primitive can test for specific conditions that must be either TRUE or FALSE. When you use an IF primitive, you must follow it with a *procedure list* — a list of instructions D.L. LOGO follows if the condition is met. A procedure list is always enclosed in square brackets.

The IF primitive works well as a test for correct answers to a quiz on addition. If the variable ANSWER contains the answer, and the numbers to sum are in the variables A and B, the IF test might look like this:

```
IF :ANSWER = :A + :B [PRINT [YOU ARE
 RIGHT!]]
```

This statement reads: If ANSWER equals the value of variable A plus variable B, then display ''YOU ARE RIGHT!'' Note that you must set square brackets around the complete procedure list as well as around the PRINT argument.

In an addition quiz, the IF primitive can also indicate that an answer is wrong:

```
IF :ANSWER <> :A+:B [PRINT [SORRY,
 YOU ARE WRONG]]
```

If the quiz answer is not equal to A+B, this command displays ''SORRY, YOU ARE WRONG.''

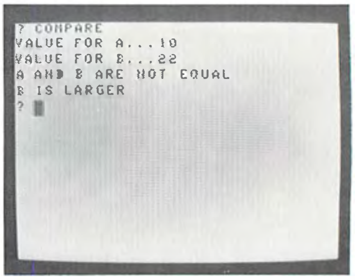
An IF statement doesn't limit you to 1 *action*. In the preceding example, you can also provide the right answer by expanding the command to:

```
IF :ANSWER <> :A+:B [PRINT [SORRY,
 YOU ARE WRONG...] PRINT [THE RIGHT
 ANSWER IS...] :A+:B]
```

Include any number of commands in the procedure list of an IF statement.

Although using IF works well in these procedures, other options are open to the LOGO programmer. The same operations using the TEST primitive look like this:

```
TEST :ANSWER=:A+:B
     IFTRUE [PRINT [THAT'S RIGHT!]
     IFFALSE [PRINT [SORRY, YOU ARE
      WRONG] PRINT [THE RIGHT ANSWER
      IS...] :A+:B]
```



```
? COMPARE
VALUE FOR A...10
VALUE FOR B...22
A AND B ARE NOT EQUAL
B IS LARGER
?
```

The IF and TEST primitives can test any statement using LOGO's comparative operators (greater-than, less-than, equal, greater-than or equal, and less-than or equal).

The IFTRUE and IFFALSE primitives are used only with the TEST primitive, and you must use 1 or both to provide output from TEST. The following procedure employs both the TEST and IF primitives in typical applications, plus the primitive ELSE in conjunction with the IF statement:

```
TO COMPARE
     MAKE "A FIRST RQ
     MAKE "B FIRST RQ
     TEST :A = :B
          IFTRUE [PRINT [A AND B ARE
           EQUAL] STOP]
          IFFALSE [PRINT [A AND B
           ARE NOT EQUAL]]
     IF :A>:B
          [PRINT [A IS LARGER]]
          ELSE [PRINT [B IS LARGER]]
END
```

Lines 1 and 2 of this procedure accept keyboard input comprising 2 numbers. The procedure then compares these numbers for equality using the TEST, IFTRUE, and IFFALSE primitives. Any number of actions contained in a procedure list can follow both the IFTRUE and IFFALSE statements. In this case, the procedure list invokes the PRINT command. If the 2 values are equal, the procedure tells you A and B are equal. If they are not equal, the IF and ELSE primitives combine to tell you which is the larger of the values.

There are no rules about when to use IF or TEST primitives. Anything that one can test, the other can test also. Using ELSE with the IF statement makes IF a preferable choice in some instances. ELSE is all inclusive; it includes every possible condition not met by the IF comparison. Thus it can save programming lines in some applications.

Feel free to use the primitive with which you feel most comfortable in any situation.

Making Selections

The SELECT primitive can also match conditions and actions. It is exceptionally neat and efficient for comparative programming. The following procedure accomplishes the same task that the IF, ELSE, TEST, IFTRUE and IFFALSE primitives perform, but only uses 1 primitive to complete the task:

```
TO COMPARE
     PRINT1 [VALUE FOR A...]
     MAKE "A FIRST RQ
     PRINT1 [VALUE FOR B...]
     MAKE "B FIRST RQ
     SELECT [
          :A=:B [PRINT [A AND B ARE
           EQUAL]]
          :A<:B [PRINT [A AND B ARE
           NOT EQUAL...B IS LARGER]]
          :A>:B [PRINT [A AND B ARE
           NOT EQUAL...A IS
           LARGER]]]
END
```

The SELECT primitive requires that you enclose all possible selections in square brackets. As well, the procedure list associated with each selection is enclosed in square brackets. As the number of brackets required in such an operation can be confusing, be sure to use an easy-to-read format that lets you follow the logic of the procedure.

The SELECT primitive processes only 1 operation. For instance, if A is greater than B, the primitive processes only the second comparison and its associated procedure list.

Testing ALL OR ANY

The primitive ALLOF can test whether all elements of a list meet specified conditions. ALLOF resembles the logical AND used with other computer languages. Using ALLOF, you can test if several conditions are all true at once. For example, you may wish to test whether a variable contains a certain value at the same time you press a certain key on the keyboard. The following procedure tests if you have pressed C when the variable R equals 12.

```
TEST (ALLOF :KEY="C :R=12)
IFTRUE [PRINT [ROW C, COLUMN 12]]
```

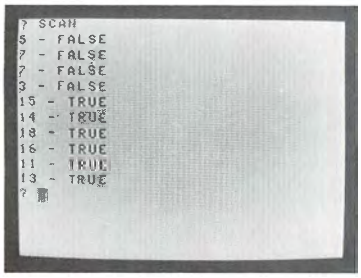
You can include as many conditions in the ALLOF comparison as you wish. You can also include other logic within the ALLOF conditions. To examine a list for a range of numbers, you can use a procedure similar to the following:

```
TEST (ALLUF :R>10 :R<20)
```

This command tests the variable R against values in the range 11 to 19. You can interpret this line to read: Test if the value of R can be found in the numbers 11 to 19. If you are already familiar with AND and OR logic, you may find it easier to read ALLOF as AND. You can then read the line as: Test if R is greater than 10 AND if R is less than 20. Because this command uses TEST, it can precede the IFTRUE and IFFALSE primitives, for instance:

```
TEST (ALLOF :R>10 :R<20)
      IFTRUE [PRINT [R IS IN RANGE]]
      IFFALSE [PRINT [R IS OUT OF
       RANGE]]
```

The ALLOF logic can expand further. To test whether a value lies between 10 and 20 but does not equal 12, use this format:



```
? SCAN
5 - FALSE
7 - FALSE
7 - FALSE
3 - FALSE
15 - TRUE
14 - TRUE
18 - TRUE
16 - TRUE
11 - TRUE
13 - TRUE
?
```

```
    TEST (ALLOF :R>10 :R<20 NOT (:R=12))
```

The following procedure puts this concept into practice:

```
    TO SCAN
         REPEAT 10 [
         MAKE "R RANDOM 20
         MAKE "M (ALLOF :R>10 :R<20 NOT
          (:R=12))
         PRINT :R [-] :M]
    END
```

When you execute the procedure, LOGO displays TRUE for all numbers between 10 and 20, except for the number 12. Also, you can store the results of comparative operations such as ALLOF in a variable. In this case, the variable M equals TRUE if the ALLOF conditions are met, or it equals FALSE if either or both of the conditions are false.

ANYOF

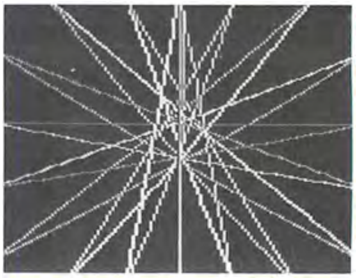
ANYOF resembles the OR command used in other computer languages. It tests to see if any value in a list meets its specified condition. For instance:

```
    TEST (ANYOF :A=:B :R>12 :M="TRUE)
         IFTRUE [PRINT [WE'VE GOT A
          MATCH]]
```

In this case, if the variable A equals the variable B, or if the variable R is greater than 12, or if the variable "M contains the word "TRUE," LOGO displays the phrase "WE'VE GOT A MATCH." ANYOF can check the Turtle coordinates in a graphics procedure such as:

```
    TEST (ANYOF XCOR>125 XCOR<-126
     YCOR>95 YCOR<-94)
         IFTRUE [PRINT [YOU ARE GOING
          OUT OF BOUNDS]]
```

A practical use of such a procedure is to return the Turtle to the screen if it goes out of bounds. The following procedure does this and, as a result, creates a graphics design:



```
TO BUMP
     HT
     CS
     FULLSCREEN
     WINDOW
     REPEAT 910 [
          CHECK
          IF :M="TRUE [RT 170 SETPC
           PC+1]
          IF PC=0 [SETPC 1]
     FD 5]
END

TO CHECK
     MAKE "M (ANYOF XCOR>124 XCOR<-
      125 YCOR>94 YCOR<-93)
END
```

In this program, the CHECK procedure uses ANYOF to see if the Turtle is at the edge of the screen. If it is, the variable M holds the value "TRUE." If M contains "TRUE," the Turtle turns right at an angle of 170 degrees and the pencolor is incremented. Because the pencolor is reset to 0 if it exceeds 3, Line 8 sets the pencolor to 1 when this happens.

A Note About NOT

In all conditional testing, LOGO also lets you use the NOT primitive. For instance, in the preceding CHECK procedure, it is valid to command:

```
TO CHECK
     MAKE "M NOT (ANYOF XCOR>125
     XCOR<-126 YCOR>95 YCOR<-94)
```

This makes M equal "FALSE" if the Turtle is at the edge of the screen. Instead of *IF :M=TRUE* in Line 8, you must write *IF :M=FALSE.*

To check whether the answer in a quiz is wrong, you can use a command like:

```
    IF NOT :ANSWER=:A+:B [PRINT [WRONG]]
```

Section 2
A Loop is a Loop is a Loop is a . . .

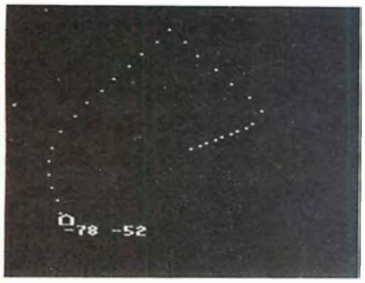
Although the REPEAT is an easy and powerful way to command your Turtle to perform a task any number of times, it does have its limitations. In many applications, it is time consuming and difficult, if not impossible, to calculate how many repetitions you need to perform. For instance, in the previous BUMP procedure, the user calculates the repeat function through trial and error. To count all the steps between the beginning and the end of the procedure is a rather odious responsibility.

WHILE Away Some Time

The primitive WHILE also creates loops. WHILE implies a DO function (WHILE some condition is true or not-true, DO a task). The following procedure uses WHILE to let you move the Turtle around the the screen until you push the joystick control button. When you push the button, the Turtle's current coordinates appear on the screen.

```
TO MOVE
     ST
     FULLSCREEN
     WRAP
     WHILE NOT BUTTON? Ø [
          SETXY XCOR+(JOYX Ø)/3
           YCOR+(JOYY Ø)/3]
     TURTLETEXT XCOR YCOR
END
```

In effect, this procedure says: While the button on joystick Ø is not pressed, increase the X and Y coordinates toward the joystick position. When the button is pressed, display the current X and Y positions.



This method of using WHILE for loop control is called *top-end* loop control because WHILE controls the loop from the top. D.L. LOGO also has a DO primitive you can use to control loops from the bottom. The following *bottom-end* loop control has the same function as the previous program:

```
TO MOVE
        ST
        FULLSCREEN
        WRAP
        DO [SETXY XCOR+(JOYX 0)/3
         YCOR+(JOYY 0)/3]
        WHILE NOT BUTTON? 0
        TURTLETEXT [XCOR YCOR]
```

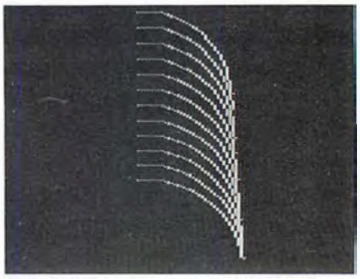
Bottom-end loop control is useful in situations where you must perform an operation at least once before you test it.

Turtle Nests

You can *nest* loops to whatever depth you wish. A nested loop is a loop within a loop. To illustrate this nesting capability, the accompanying procedure tests 2 conditons: the Y coordinate and the Turtle heading.

```
TO ARCS
CS HT
MAKE "Y -20
WHILE :Y<100 [
        SETXY -30 :Y
        SETHEADING 90
        WHILE HEADING <175 [
                FD 5 RT :Y/95+8]
        MAKE "Y :Y+10]
END
```

This procedure draws a series of arcs, each above the other. The first WHILE loop executes until the variable Y is greater than 100. The second WHILE loop executes until the Turtle's heading is less than 165.



FOR in a loop

FOR resembles the REPEAT primitive, except it directly manipulates a variable. This manual has already shown several procedures that use a counter in a loop, increasing or decreasing it each time the loop executes. For example:

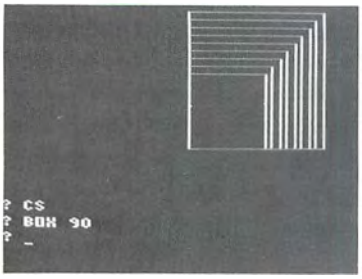
```
TO BOXUP
     MAKE "COUNT 1
     REPEAT 10 [REPEAT 4 [
            FD :COUNT RT 90]
          MAKE "COUNT :COUNT+10]
END
```

The FOR primitive can make the procedure much shorter:

```
TO BOXUP
     FOR "COUNT 1 100 10 [
          REPEAT 4 [FD :COUNT RT
           90]]
END
```

The FOR primitive has 5 arguments or parameters:

- The name of the variable that stores the index
- The initial index (variable) value
- The final index (variable) value
- The rate of increment or decrement (the step rate)
- The procedure list to execute



```
? CS
? BOX 90
? _
```

The logic of the preceding procedure is: set COUNT to 1 and increment to 100 using a step rate of 10. The first time through the FOR loop, COUNT equals 1; the second time through the loop, COUNT is 11, the third time, COUNT is 21; and so on.

You can also use FOR in decreasing loops. You can produce the same design using FOR in the following manner:

```
TO BOXDOWN
     FOR "COUNT 91 1 -10 [
          REPEAT 4 [FD :COUNT RT 90]
     ]
END
```

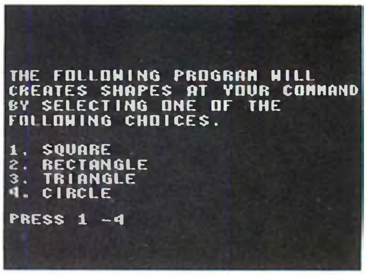
In this case, the variable COUNT is set at 91 and decreased by 10 in each loop.

On the Go

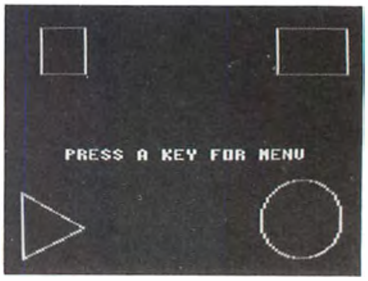
At times it is important to be able to repeat a portion of a procedure. By using the primitives LABEL and GO, you can select points of execution in any size procedure. For instance, the following procedure lets you choose a shape to create without erasing previously created shapes from the graphics screen or reprinting the initial instructions:

```
TO SHAPE
     CS
     CLEARTEXT
     SETSPLIT 12
```

```
        PRINT [THE FOLLOWING PROGRAM
         CREATES]
        PRINT [SHAPES AT YOUR COMMAND
         WHEN]
        PRINT [YOU SELECT ONE OF THE]
        PRINT [FOLLOWING CHOICES.]
        PRINT []
        SPLITSCREEN
        PRINT [1. SQUARE]
        PRINT [2. RECTANGLE]
        PRINT [3. TRIANGLE]
        PRINT [4. CIRCLE]
        PRINT []
        PRINT1 [PRESS 1-4....]
    LABEL "NEXT
        MAKE "CHOICE RC
        FULLSCREEN
        SELECT [
            :CHOICE=1 [SQUARE]
            :CHOICE=2 [RECTANGLE]
            :CHOICE=3 [TRIANGLE]
            :CHOICE=4 [CIRCLE]
        SETXY -80 0
        TURTLETEXT [PRESS A KEY FOR
         MENU]
        MAKE "C RC
        SETSPLIT 8
        SPLITSCREEN
    GO "NEXT
END

    TO SQUARE
        SETXY -95 50
        REPEAT 4 [FD 30 RT 90]
    END

    TO RECTANGLE
        SETXY 60 50
        REPEAT 2 [FD 30 RT 90 FD 50 RT
         90]
    END
```



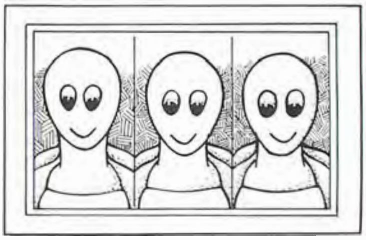
```
THE FOLLOWING PROGRAM WILL
CREATES SHAPES AT YOUR COMMAND
BY SELECTING ONE OF THE
FOLLOWING CHOICES.

1. SQUARE
2. RECTANGLE
3. TRIANGLE
4. CIRCLE

PRESS 1 -4
```



```
PRESS A KEY FOR MENU
```

```
TO TRIANGLE
     SETXY -110 -80
     REPEAT 3 [FD 50 RT 120]
END

TO CIRCLE
     SETXY 50 -50
     REPEAT 180 [FD 1 RT 2]
END
```

Section 3
A Recursive Turtle



One way of controlling the duration of a loop is simply to stop the procedure when it reaches a certain condition. This method works well in a *recursive* procedure (a procedure that calls on itself). A simple recursive procedure is:

```
TO DISPLAY
     PRINT [HELLO]
     DISPLAY
END
```

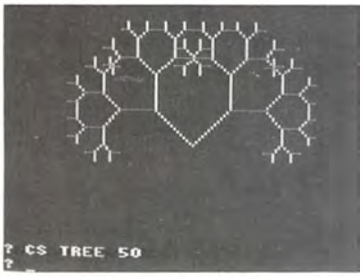
If you execute this procedure the screen displays "HELLO" repeatedly until you press BREAK or turn off your computer. To control such a procedure, use the REPEAT primitive or some kind of conditional statement that stops the procedure when it meets its condition. A conditional statement with a variable counter is 1 method:

```
TO DISPLAY :COUNT
     MAKE "COUNT :COUNT+1
     IF :COUNT>10 [STOP]
     PRINT [HELLO]
     DISPLAY :COUNT
END
```

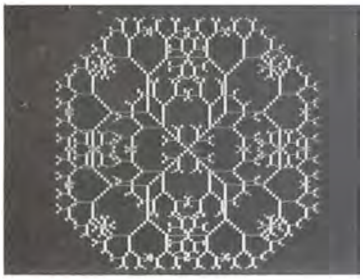
To execute this procedure, you must provide a value for COUNT when you execute the program, such as:

```
? DISPLAY 1 ENTER
```

Providing a value of 1 causes DISPLAY to display "HELLO" 10 times. Although such a procedure is easy to understand, it is not very useful. Recursive procedures with conditional control can be more impressive. The following program uses such a recursive technique to produce a binary tree. (Each branch has 2 appending



```
? CS TREE 50
?
```

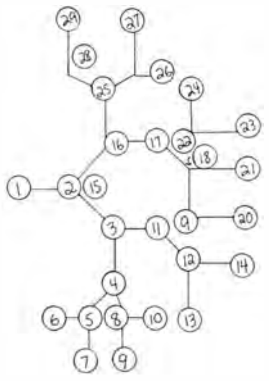


branches.) After you type and execute the program, imagine writing it without using any kind of loops.

```
    TO FLAKE
         CS
         FULLSCREEN
         TREE
         RT 180
         TREE
    END

    TO TREE
         BI 17
         BI 10
         BI 6
    END

    TO BI :LENGTH
         IF :LENGTH<3.5 [STOP]
         RT 45
         FD :LENGTH*2
         BI :LENGTH*.74
         BK :LENGTH*2
         LT 90
         FD :LENGTH*2
         BI :LENGTH*.74
         BK :LENGTH*2
         RT 45
    END
```

The recursive portion of this program is in the BI procedure. First, the procedure causes the Turtle to turn right 45 degrees and draw a line. To create a full branch on the tree the program repeats this procedure, using smaller branches each time. The BI procedure does this by calling on itself to create a new twig on the end of the preceding one, until it reaches a size less than 3.5. The STOP primitive ends the recursive loop at this point.

The procedure now begins executing the lines that follow the command to reexecute itself. It proceeds backward until it reaches a junction point of a left branch. In this case, it is the smallest branch. It draws this branch and then returns to create the next left branch. This operation continues until the procedure reaches the original starting point and completes a full branch. Then the procedure begins on the left side of the original branch and builds it in a similar manner. The accompanying diagram numbers each branch in the order it is drawn.

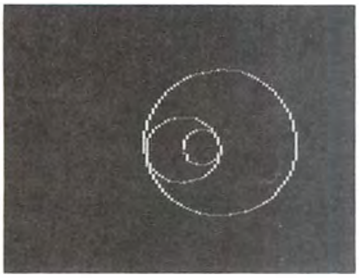


If the recursive concept is difficult to understand in the previous example, you may find the following procedure easier:

```
TO CIR :AMT
     FULLSCREEN
     IF :AMT <5 [STOP]
     REPEAT 180/:AMT [FD 30/:AMT RT
       :AMT]
     CIR :AMT*.7
     REPEAT 180/:AMT [FD 30/:AMT RT
       :AMT]
END
```

To execute this procedure, you must provide a value for AMT. If the value is too low, Line 1 of the procedure causes the operation to stop before anything happens. A value between 5 and 7 produces a single circle. A value of 8 produces a circle within a circle. A value of 11 produces a circle within a circle within a circle.

The procedure directs the Turtle to draw a circle, a half-circle at a time. A recursion call is issued after the Turtle draws the first half-circle. This draws another half circle, larger than the first. As it draws successive half-circles, the Turtle spirals outward. When AMT decreases (due to multiplying by 0.7) to a value less than 5, the recursive calls stop. While the Turtle draws each half-circle, D.L. LOGO stores the portion of the repetitive procedures that come after the recursive calls in a "last in, first out" manner. Now that all the initial half-circles are drawn, the operation recalls the last portion of the commands



and draws all the second half-circles, causing an inward spiral that ends when the original circle is complete.

A procedure can use recursive calls more than once. You can expand the CIR procedure to include 2 recursive calls:

```
TO CIR2 :AMT
     IF :AMT <5 [STOP]
     REPEAT 180/:AMT [FD 25/:AMT RT
       :AMT]
     CIR2 :AMT*.7
     REPEAT 180/:AMT [FD 25/:AMT RT
       :AMT]
     CIR2 :AMT*.7
END
```

This procedure works in the same manner as the previous CIR procedure; however, the procedure issues 2 recursive calls, and the Turtle draws circles to create an overlapping mirror-image.

Chapter Summary

PRIMITIVE	Abbrev.	Purpose
IF	–	A condition test. Conditionally executes a procedure list.
ELSE	–	Runs a procedure list if the preceding IF test fails.
TEST	–	A condition test. Sets a condition register for subsequent IFTRUE or IFFALSE procedures.
IFTRUE	–	Runs a procedure list if the condition register (set by TEST) contains TRUE.
IFFALSE	–	Runs a procedure list if the condition register (set by TEST) contains FALSE.
SELECT	–	Executes the procedure list of the first subsequent TRUE condition.
WHILE	–	Establishes a top-end control loop. Executes a procedure *while* a specified condition is true.
DO	–	Establishes a bottom-end control loop. Repeats the execution of a procedure list as long as the subsequent WHILE function is TRUE.

PRIMITIVE	Abbrev.	Purpose
FOR	–	Performs an automatically increasing index for loop control.
LABEL	–	Flags a position in a procedure for the GO function.
GO	–	Sends the execution of a procedure to the indicated LABEL position.

Turtle Facts

- Conditional logic and loop primitives extensively work together in LOGO.
- A procedure list following a conditional command is always enclosed in square brackets.
- Procedure lists following conditional commands can contain multiple commands or actions.
- You can nest loops to any depth your computer's memory can handle.
- You can use recursive calls as many times as you wish within the limits of your computer's memory.

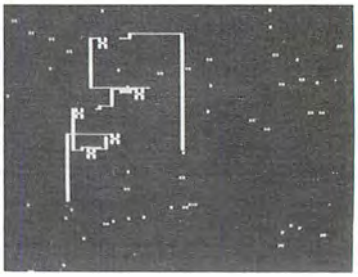
Suggested Project

Write a program that randomly places 100 dots on the screen. Then, use the 4 arrow keys to direct a cursor to hit the dots. Make the game last about 50 seconds and provide a score of the number of dots hit before the game ends.

Suggested Project Solution

The Game of Seek

```
TO SEEK
     DO[
     PLAY
     FINISH
     SPLITSCREEN
     PRINT [DO YOU WISH TO PLAY AGAIN?]
     MAKE "CHOOSE RQ]
     WHILE (ANYOF :CHOOSE=[YES]
      :CHOOSE=[Y] :CHOOSE=[OK] :CHOOSE=[
      YEP])
END

TO PLAY
     HT
     CS
     FULLSCREEN
     MINE
     WRAP
     MAKE "HIT 1
     MAKE "COUNT 0
     SETPC 3
     HOME
     INPUT
END

TO INPUT
     WHILE :COUNT<500 [
          MAKE "COUNT :COUNT+1
          FD 1
          CHECK
          IF KEY? [READKEY]
          ]
END
```



```
TO READKEY
     MAKE "K RC
     SELECT [
          :K=CHAR 12 [SETH 0]
          :K=CHAR  9 [SETH 90]
          :K=CHAR 10 [SETH 180]
          :K=CHAR  8 [SETH 270]
          :K=CHAR  3 [END]
     ]
END

TO CHECK
     MAKE "T LIST XCOR YCOR
     IF MEMBER? :T :COR [XXX]
END

TO MINE
     SETPC 1
     MAKE "COR LIST "
     REPEAT 60 [
     MAKE "X (RANDOM 250)-125
     MAKE "Y (RANDOM 190)-95
     MAKE "L LIST :X :Y
     MAKE "COR SENTENCE LPUT :L :COR
     SETDOT]
END

TO SETDOT
     SETXY :X-1 :Y-1
     REPEAT 4 [FD 1 RT 90]
END

TO XXX
     TURTLETEXT [X]
     SAY [HIT NUMBER] :HIT
     MAKE "HIT :HIT+1
END
```

```
TO FINISH
     SETSPLIT 8
     SPLITSCREEN
     PRINT [NUMBER OF HITS=]:HIT-1
     SAY [THIS GAME IS OVER]
END
```

NOTE: If you have the Speech-Sound Cartridge, this game keeps a verbal tally of the hits you make and informs you when the game ends. Otherwise, the game does not produce verbal responses.

11 TALKING BACK

Turtle Talk, Chatter, and Other Noise

Section 1 You Write the Speech: telling Turtle what to say.
Section 2 A Sound Procedure: Turtle sounds off.

Section 1
You Write The Speech

Note: This section is only for those who have the Speech/ Sound Cartridge and the Multi-Pak Interface. You need both these items to use D.L. LOGO's speech capabilities. The SOUND primitive, described in Section 2, does not require the Speech/Sound cartridge.

Setting Up

D.L. LOGO's speech capability is exciting, whether you are experimenting and writing procedures for yourself or writing programs for others. The ease in adding speech to your procedures makes this feature even more appealing.

Before attempting the examples in this section, be sure you connect the Multi-Pak Interface to your computer and correctly insert both the disk and sound/speech cartridges into the Multi-Pak slots.

SAY, Turtle Can Talk

To make your Turtle talk, use the primitive SAY. For example, to have the Turtle say the word HELLO, turn up the volume on your television set and type:

```
? SAY [HELLO] ENTER
```

D.L. LOGO immediately responds by generating the word HELLO through your television speaker. Try other words and sentences. There is no limit to what your Turtle can say. Remember to enclose the words inside the square brackets. For example, type:

```
? SAY [HOW ARE YOU ON THIS FINE
DAY?] ENTER
```

A Little Speech Therapy

Some of the words D.L. LOGO pronounces do not sound quite right. Because of the peculiarities of English pronunciation, it is impossible for the speech/sound cartridge to correctly pronounce all the words you give it. Sometimes the program needs extra help. To illustrate this, ask LOGO to say the word COUNTRY by typing:

```
? SAY [COUNTRY] ENTER
```

The word sounds like KOWNTREE.

Now type:

```
? SAY [KUNTRY] ENTER
```

Many words need phonetic spelling in order for the Speech/Sound Cartridge to handle them properly. Often, several spellings work equally well. For COUNTRY, try:

```
? SAY [CUNTREE KUNTRY] ENTER
```

The 2 spellings sound the same.

Type and Talk

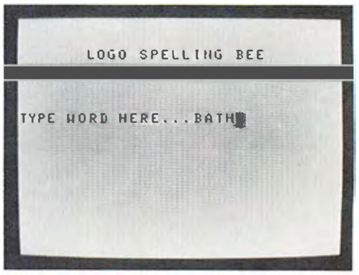
You may want to try other word and sound combinations without typing the primitive SAY each time. To do this, type the following procedure:

```
TO TALK
WHILE "TRUE [SAY RQ]
END
```

Now, exit the Edit mode and type:

```
? TALK ENTER
```

Type any words or sentences, and then press ENTER. D.L. LOGO says whatever you type. Use this program to try different spellings, and make notes of your results for future reference. To exit the procedure, press BREAK.


```
LOGO SPELLING BEE

TYPE WORD HERE...BATH
```

Turtle Teaches Spelling

You can use a talking Turtle in many ways. The following demonstration program shows how this talking ability can teach spelling.

Many computer-aided spelling programs require students to choose correct spellings from lists of correctly and incorrectly spelled words. Rather than reinforcing proper spelling, this often confuses a student by making it hard to remember which of the several spellings on the screen is right. Turtle's ability to speak the words that the student spells provides positive reinforcement of the correct spelling.

```
TO SPELL
     CLEARTEXT
     SETUP
     SETCURSOR 2 7
     PRINT [LOGO SPELLING BEE]
     PRINT :DIV
     ASK
END

TO ASK
     FOR "T 1 COUNT :WORDS 1
          [MAKE "WORD ITEM :T :WORDS
          SETCURSOR 6 0 B 32
          PRINT [LISTEN FOR NEXT
           WORD]
          WAIT 100
          SETCURSOR 5 2 B 29
          SAY [HOW DO YOU SPELL]
           :WORD
          SETCURSOR 7 0 B 32
          SETCURSOR 6 0
          PRINT1 [TYPE WORD HERE...]
          B 10 CHECK]
     SETCURSOR 10 4
     PRINT [THAT IS ALL...]
END

TO SETUP
     MAKE "WORDS [CAT BOOK HAT KEY
      WATER MILK PAPER BIRD HELP
      BATH RAIN]
     MAKE "BLK "\ \ \ \ \ \ \ \ \ \
      \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
      \ \ \ \ \ \ \ \ \ \
     MAKE "DIV "ixixixixixixix
      ixixixixixixixixix
END

TO CHECK
     MAKE "ANSWER FIRST RQ
     SELECT [
         :ANSWER=:WORD [
          SAY [THAT IS CORRECT]]
         :ANSWER <> :WORD [
          SAY [YOU ARE WRONG]
          PRINT1 [THE RIGHT
           SPELLING IS...\ ]
           :WORD
     ]
     END

TO B :SPACE
     MAKE "BLANK PIECE 1 :SPACE :BLK
     MAKE "L LINE
     MAKE "C COLUMN
     PRINT1 :BLANK
     SETCURSOR :L :C
END
```

Section 2
A Sound Procedure

The SOUND primitive rounds out an extremely varied repertoire of audio responses. With speech, music, and sound at your command, you can expand D.L. LOGO's programming to the limits of your imagination. With these capabilities, you can:

- Write games with sound, speech, and a musical introduction
- Instruct your Turtle to tell you where it is on the screen, what it is doing, and how long it is taking
- Create sounds for musical instruments or turn your computer keyboard into a music keyboard
- Create a talking alarm clock
- Write instructional programs

These are only a few of the many possibilities that sound, music, and speech offer D.L. LOGO. Although there is only 1 new primitive to learn in this section, take your time and experiment with SOUND. D.L. LOGO makes coaxing fascinating sounds from microcomputer chips easy and fun.

Making Sound

The SOUND primitive requires 2 arguments: pitch and duration. The pitch is represented in cycles per second, and the duration is represented in milliseconds. If this means nothing to you, don't worry. You can create exciting sounds through experimentation without knowing anything about cycles or milliseconds. Although the cycle argument has a range of −32767 to 32767, not everything in this range is practical to use. For instance, the negative range produces the same results as the positive range. Also the values between 7000 and 26000 give poor results. The following program demonstrates how notes sound in all the ranges.

```
TO RANGE
     FOR "I -32767 32767 100 [
          SOUND :I 50]
END
```

The procedure uses a FOR loop, adding increments of 100 cycles to each step, to demonstrate sounds throughout the SOUND range. Notice that the direction of the pitch changes several times during the demonstration and that most of the midsection of the negative and positive ranges is inaudible. You can, in fact, hear all the possible pitches by using values between 100 and 7000.

It is easy to calculate the length of a specified duration value. Because there are 1000 milliseconds in a second, a duration of 1000 lasts approximately 1 second.

Using Repeat

You can use several methods to create different types of sounds. One way is to use REPEAT to produce a sound in rapid succession. The following procedure demonstrates this:

```
TO BOMPA
     REPEAT 30 [SOUND 350 10]
END
```

A short "bomp" sound is produced. What sounds can you make by stringing several "bomp" sounds together? Try it with:

```
TO BOMPABOMP
     REPEAT 10 [BOMPA WAIT 10]
END
```

Perhaps you can create some different "bomp" sounds. Try the following:

```
TO BOMP1
        REPEAT 12 [SOUND 350 7]
END

TO BOMP2
        REPEAT 20 [SOUND 350 9]
END

TO BOMP3
        REPEAT 10 [SOUND 350 8]
END

TO BOMP4
        REPEAT 40 [SOUND 350 12]
END
```

Produce different sounds by slightly changing the pitch and duration values. If you like a "bomp" rhythm, try this:

```
TO BOMP
        BOMP1
        WAIT 2
        BOMP1
        WAIT 2
        BOMP1
        WAIT 5
        BOMP2
        WAIT 8
        BOMP3
        WAIT 5
        BOMP4
END
```

Or, add some new "bomp" values and try another rhythm:

```
TO BOMP5
        REPEAT 30 [SOUND 350 12]
```

```
END

TO BOMP6
        REPEAT 8 [SOUND 350 15]
END

TO BOMP7
        REPEAT 15 [SOUND 350 10]
END

TO BOMP8
        REPEAT 30 [SOUND 350 10]
END

TO BOMP9
        REPEAT 25 [SOUND 350 6]
END

TO TWOBOMP
        BOMP5
        WAIT 20
        BOMP6
        WAIT 5
        BOMP7
        WAIT 30
        BOMP6
        WAIT 5
        BOMP7
        WAIT 15
        BOMP8
        WAIT 3
        BOMP9
        WAIT 17
        BOMP6
END
```

The following program uses the REPEAT procedure to create the sound of a ringing phone:

```
TO PHONE
     REPEAT 3 [
          REPEAT 70 [SOUND 1000 10]
          WAIT 150]
     REPEAT 3 [SOUND 1000 10 WAIT 5]
     SOUND 800 5 SOUND 750 550
     REPEAT 10 [SOUND 750 250 WAIT
      20]
END
```

For Making Sound

The FOR loop is also an efficient way to create modulated sounds. The following SOUND procedures all use FOR loops.

```
TO LAZER
     FOR "I 5000 200 -100
          [SOUND :I 3]
     END

TO ZAP
     FOR "I 1000 6000 100
          [SOUND :I 1]
END

TO SPACE
     REPEAT 10
          [FOR "I 5000 1000 -1000 [
            SOUND :I 20]]
     REPEAT 4
          [FOR "I 1000 10000 500 [
            SOUND :I 10]]
END
```

Sounding Off with Math

Although you can include calculations in your sound routines, they often slow the process too much to create the kind of sound you want. You can bypass this problem by creating a list of calculation results and then *playing* the list. For instance, the following procedure creates a list of values produced by the TAN function. The ITEM primitive selects 1 value at a time for the SOUND primitive. Because this method is much faster than including the calculations in the SOUND routine, the sound resembles a buzzing fly.

```
TO FLY
    MAKE "I []
    FOR "N 0 81 1 [PRINT :N
    MAKE "I LPUT
    5000+4000*(TAN:N) :I]
    LABEL "X
    MAKE "X1 RC
    FOR "N 1 82 1 [
    SOUND ITEM :N :I 5]
    GO "X
END
```

The procedure counts the items as it builds the list. When it finishes counting, it waits for you to press a key. Then, it demonstrates the sound. You can replay the sound any number of times by pressing a key again. For a different sound, use FPUT rather than LPUT.

On Your Own

Don't be content with the sounds demonstrated in this manual. Strike out on your own and create different sounds in different ways. Make a library of procedures you can use in later applications. You can also use D.L. LOGO's music capabilities, described in Chapter 6, to create unusual sounds.

Sound Speech

If you have the Speech/Sound Cartridge, you can also use speech to create unusual sounds with ease. To demonstrate this, type and execute the following lines:

```
? SAY [UUUUUUUUUUUUUUUUUUUUUUUU
  UUUUUUUUUUUUUU] ENTER
? SAY [SSSSSSSSSSSSSSSSSSSSSSSS
  SSSSSSSSSSSSSS] ENTER
? SAY [SHSHSHSHSHSHSHSHSHSHSHSH
  SHSHSHSHSHSHSHSHS] ENTER
? SAY [BZZZ] ENTER
? SAY [RSRSRSRSRSRSRSRSRSRSRSRS
  RSRSRSRSS] ENTER
? SAY [SYSYSYSYSYSYSYSYSYSYSYSY
  SYSYSYSYSYSYSS] ENTER
? SAY [OPOPOPOPOPOPOPOPOPOPOPOP
  OPOPOPOP] ENTER
? SAY [QQQQQQQQQQQQQQQQQQQQQQQQ
  QQQQQQQQQQQQ] ENTER
? SAY [MNMNMNMNMNMNMNMN]
```

Try other letters and combinations. Turtle has a voice with many talents.

Chapter Summary

PRIMITIVE	Abbrev.	Purpose
SAY	–	Causes D.L. LOGO to say (speak) the specified words or numbers.
SOUND	–	Creates a sound from specified values representing pitch and duration.

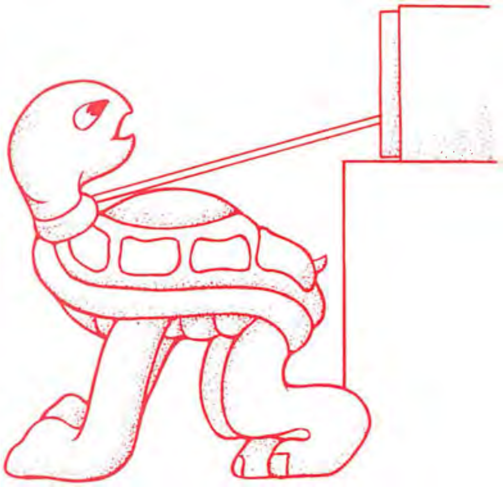
Turtle Facts

- You must have a Speech/Sound Cartridge and a Multi-Pak Interface to use D.L. LOGO's speech ability
- You need to spell some words phonetically in order for the Speech/Sound Cartridge to pronounce them correctly
- Using the SOUND primitive, pitch is expressed in cycles per second and duration is expressed in milliseconds

Suggested Project

Write a short, educational program about a subject that is familiar to you. Our sample program is the DEMO program that is included on your D.L. LOGO diskette.

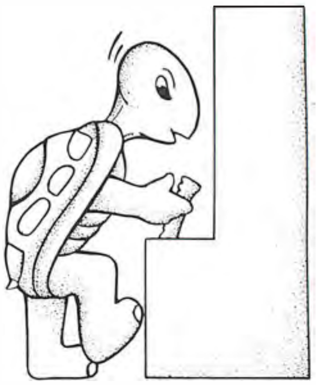
12 TURTLE ON A LEASH



Joystick and X-Pad Controls

Section 1	Reading the Joysticks: put a handle on Turtle.
Section 2	Padding About Turtle Style: using the X-Pad graphics tablet.
Section 3	Remote Control: information on controlling remote devices.

Section 1
Reading the Joysticks



The fun of D.L. LOGO doesn't end with impressive screen displays. Your Turtle can respond to input from sources other than the keyboard, such as joysticks or an X-Pad. Through D.L. LOGO, you can control external devices, such as a mechanical turtle, a robot, or a plotter.

These options create numerous new possibilities for LOGO. For instance, if you were to write an educational program for young children, you can let them make choices using the joystick instead of keyboard characters. You can create art and music programs using an X-Pad. You can send graphics designs to a plotter or printer to create pictures or posters.

The joystick and X-Pad controls are built into D.L. LOGO and are easy to use. In order to use the X-Pad with D.L. LOGO, however, you must have the Multi-Pak Interface.

To control external devices, such as a robot or plotter, you must have a control module created specifically for that device. Section 4 provides the information necessary for creating such a machine-language module and is only of value to advanced machine-language or assembly-language programmers.

Taking Control

D.L. LOGO uses numbers to represent the left and right joysticks. To read the screen location of the right joystick, use this command syntax:

```
    JOYX Ø  JOYY Ø.
```

To check on how this primitive works, be sure you have a joystick plugged into the right joystick port and type:

```
? PRINT JOYX 0 JOYY 0 ENTER
```

Depending on the position of the joystick handle, the screen shows 2 values ranging between −32 and +32. Refer to the information in Chapters 1 and 4 to refresh your memory on how the graphics screen is divided into coordinates.

JOYX 0 returns the *X* coordinate of the right joystick. JOYY 0 returns the Y coordinate of the right joystick. Similarly, JOYX 1 and JOYY 1 represent the *X* and *Y* coordinates of the left joystick. The value of the joystick X and Y readings range from −32 to 31.

The following procedure reads the joystick coordinates and displays a dot on the screen in that location. It continues doing so until you press the red fire button on your joystick.

```
TO JOY
    WHILE NOT BUTTON? 0
    [DOT JOYX 0 JOYY 0 ]
END
```

When you move the joystick handle the screen displays a dot that corresponds to the position of the handle. The dot's range, however, is restricted to an area within 32 steps in any direction from the center of the screen.

Use the following procedure to view the actual values as you manipulate the joystick:

```
TO JOY2
    WHILE NOT BUTTON? 0 [
        PRINT JOYX 0 JOYY 0]
END
```

For most graphics applications, a range of only 64 steps in any direction is unsatisfactory. To expand the range of your Turtle on a leash, add these steps to your procedure:

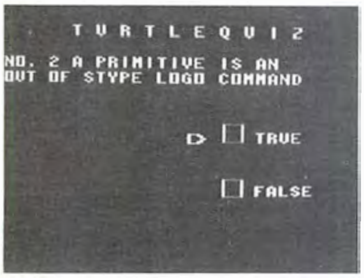
```
TO JOY3
     FULLSCREEN
          WHILE NOT BUTTON? 0
          [DOT (JOYX 0)*4 (JOYY
            0)*3]
END
```

Now you can create dots 4 steps apart on the X axis and 3 steps apart on the Y axis over the entire screen. In some applications this is satisfactory. To create dots at all points, however, use a procedure such as this:

```
TO JOY4
     FULLSCREEN
          WHILE NOT BUTTON? 0 [
          SETXY XCOR+(JOYX 0)/5
            YCOR+(JOYY 0)/5
          DOT XCOR YCOR]
END
```

In this procedure, the joystick controls the direction of increment. Thus, if you position the joystick in the positive portion of the X axis, the screen displays dots toward the right. If you position the joystick in the negative portion of the X axis, the screen displays dots toward the left. The same process holds true for the position of dots along the Y axis. In effect, the dots follow the direction of the joystick handle.

Now the Turtle can move anywhere on or off the screen. To make the Turtle reappear on the opposite side of the screen when it goes off one edge, type **WRAP** ENTER.

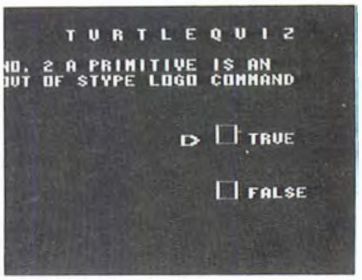


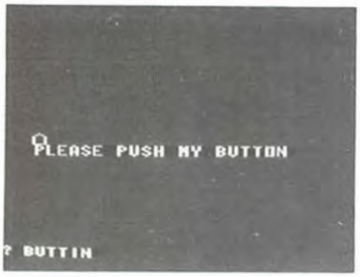
Joystick Answers

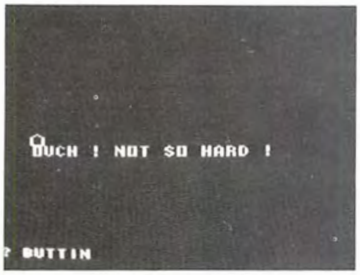
The following procedure shows how you can use the joystick in a quiz program:

```
TO QUIZ
    WRAP
    MAKE "I 0
    QUE
    FULLSCREEN
    ASK
END

TO ASK
    WHILE NOT EMPTY? :QUESTIONS
    [CS
        GET
        MAKE "I :I+1
        SHOWQUES
        T&F
        JOY
        READANS
        SETXY -40 -60
        HT
        TELLANS
        WAIT 100
        ST ]
END

TO BOX
    REPEAT 4 [FD 12 RT 90]
    SETH 90 PU FD 20 PD
END

TO DELETE
    MAKE "QUESTIONS BUTFIRST
     :QUESTIONS
END
```

```
TO JOY
     WHILE NOT BUTTON? 0 [
     SETXY XCOR+(JOYX 0)/5
      YCOR+(JOYY 0)/5 ]
END

TO QUE
     MAKE "QUESTIONS [[A JOYSTICK IS
      HAPPY GLUE][][FALSE][A
      PRIMITIVE IS AN][OUT OF STYLE
      LOGO COMMAND][FALSE][A
      PROCEDURE IS AN][ACCEPTABLE
      WAY TO GET A DATE][FALSE][JOYX
      IS A COORDINATE][POSITION][
      TRUE]]
END

TO GET
     MAKE "Q FIRST :QUESTIONS DELETE
     MAKE "Q1 FIRST :QUESTIONS
      DELETE
     MAKE "A FIRST :QUESTIONS DELETE
END

TO SHOWQUES
     SETXY -80 80
     TURTLETEXT [T U R T L E
      Q U I Z]
     SETXY -122 YCOR-20 TURTLETEXT [
      NO.] :I :Q
     SETY YCOR-10  TURTLETEXT :Q1
END

TO READANS
     SELECT [
          YCOR>0 [MAKE "AN [TRUE]]
          YCOR<0 [MAKE "AN [FALSE]]
     ]
END
```


```
T U R T L E Q U I Z
NO. 2 A PRIMITIVE IS AN
OUT OF STYPE LOGO COMMAND

D □ TRUE

  □ FALSE
```





```
TO TELLANS
     SELECT [
          :AN=:A [TURTLETEXT [YOU
           ARE RIGHT!]]
          :AN<>:A [TURTLETEXT [YOU
           ARE WRONG!]]]
END

TO T&F
     SETXY 20 5 BOX
     SETY 11 TURTLETEXT [TRUE]
     SETXY 20 -21 BOX
     SETY -26 TURTLETEXT [FALSE]
END
```

In this program, the JOY procedure lets you position the Turtle over the appropriate TRUE or FALSE prompt and press the joystick button to register your response.

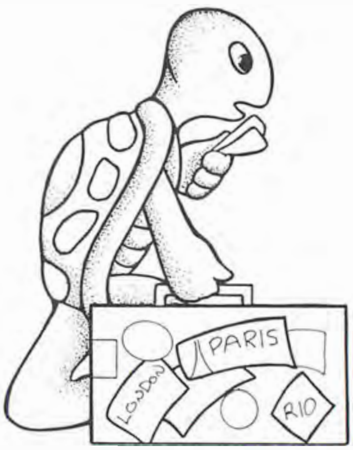
Buttoning Up the Joystick Commands

The previous joystick examples familiarized you with the BUTTON? primitive. The BUTTON? primitive determines if you press the joystick button. As with the JOYX and JOYY primitives, you need to tell BUTTON? what joystick to examine. Again, use the number 0 for the right joystick and the number 1 for the left joystick.

This procedure demonstrates another way to use the BUTTON? primitive:

```
TO BUTTIN
     CS
     SETXY -100 0
     TURTLETEXT [PLEASE PUSH MY
      BUTTON]
     WHILE BUTTON? 0 [TURTLETEXT [
      OUCH! NOT SO HARD!]]
     BUTTIN
END
```

Section 2 Padding About - Turtle Style



Using an X-Pad Graphic Tablet requires only 3 primitives: PADX, PADY, and PADPENDOWN?. You can use these primitives with other D.L. LOGO functions to create numerous uses for the X-Pad. You use the X-Pad most often for drawing, and a simple program lets you create cartoons, schematics, designs, or whatever you wish. The possibilities for the X-Pad are limitless. For instance, you can use your knowledge to program:

- A talking clock—use the X-Pad to set a time, and D.L. LOGO speaks the hour and minute
- A music board—play notes by touching corresponding keys drawn on the X-Pad
- A moving dot game—try to catch a dot on the screen using the pen on the X-Pad
- A noise board—produce various sounds by touching various points on the X-Pad.
- A battleship game—sink the enemy ship by guessing where it is on the X-Pad grid

Of course, this versatile peripheral provides many other options. Children, in particular, like the control and instant response that an X-Pad gives.

Pen Control

Plug the X-Pad into any port of the Multi-Pak Interface. D.L. LOGO sets the X-PAD grid to match your screen grid. Therefore, the position of the pen on the X-PAD always matches a corresponding position on the screen. The X coordinates are in the range −96 to 95, and the Y

coordinates are in the range −192 to 191. Read the position of the X-Pad pen by using the PADX and PADY primitives.

Programming for the X-Pad is similar to programming for the joysticks. Anything you can do with a joystick, you can do with the X-Pad. However, because the X-Pad grid is larger and the X-Pad board provides a flat surface, you can accomplish detailed tasks easier than you can with a joystick.

To see how you control the X-Pad, try this procedure:

```
TO DRAW
     CS
     WHILE "TRUE[
     WHILE PADPENDOWN?
          [SETXY PADX PADY FD 1]]
END
```

Now you can draw on the X-Pad and create corresponding graphics on the display screen. To round out the program and provide additional features, use the template holes for establishing pen position coordinates. Then use these template holes to call certain functions. To determine the pen coordinates of the various template positions, use the following procedure:

```
TO POSITION
     LABEL "AGAIN
     TEST PADPENDOWN?
          IFTRUE [PRINT PADX PADY]
     GO "AGAIN
END
```

Now, whenever you press the pen down, the screen displays its current position and you can record the coordinates you wish to use in a program. Be sure you do not have the FULLSCREEN option set when you use this procedure.

Using The Template

The following program uses some of the template positions to add features to the DRAW procedure. To operate it, place a sheet of paper on the X-Pad and use the overlay template. The X-Pad reproduces the drawings on the paper onto the screen.

```
TO PAD
     FULLSCREEN
     HT
     LABEL "PADD
     WHILE PADPENDOWN? [SETXY
          PADX PADY FD 1]
     IF (ALLOF PADX>116 PADY>93) [CS
      HT]
     IF (ALLOF PADX>112 PADX<116
      PADY>86) [SOUND 600 100]
     IF (ALLOF PADX>-6 PADX<3 PADY
      >94) [SETPC 0 ST]
     IF (ALLOF PADX<-55 PADX>-62
      PADY>94) [SETPC 3 HT]
     IF (ALLOF PADX<-114 PADY>85
      PADY<94) [SETPC 0 BOX]
     IF (ALLOF PADX<-114 PADY>68
      PADY<75) [SETPC 1 BOX]
     IF (ALLOF PADX<-114 PADY>48
      PADY<58) [SETPC 2 BOX]
     IF (ALLOF PADX<-114 PADY>30
      PADY<38) [SETPC 3 BOX]
     GO "PADD
END

TO BOX
     SETXY 124 93
     REPEAT 4 [FD 1 RT 90]
END
```

This program provides the following functions:

1. Draws when the pen is down

2. Changes pen color by touching the pen at template positions 0, 1, 2, or 3

3. Clears the screen by touching the pen at template position CLEAR

4. Provides an erase feature, with the Turtle visible, by touching template position ERASE

5. Shows the current pen color by providing a spot of color in the top right corner of the screen.

6. Produces a sound when the pen touches template position *a*.

You can easily expand this program to recognize many other features, such as the ability to: draw predetermined shapes, change background colors, create various sound effects, display text on the screen, and more.

Section 3
Remote Control

This section is for machine-language or assembly-language programmers. It provides the necessary information for creating a special module that transfers Turtle commands to an external device. If you are not an advanced machine-language or assembly-language programmer, this section is not useful to you.

Creating An Interface Module

D.L. LOGO has a special hook that can intercept Turtle's graphics commands and use them to control such devices as an external Turtle, plotter, or robot. If a module named ''Turtle'' is loaded into OS-9 memory before LOGO is executed, Turtle's graphics commands are routed through that module as a subroutine instead of moving the turtle on the screen. The module must relay each instruction to the auxiliary device and return to LOGO using an RTS instruction.

When you enter the subroutine, the A register holds the command code and the X and U registers hold any parameters associated with the command. Upon exiting the subroutine, commands that return data (shown in the following table) use the X and U registers.

		Registers			
		Entry		Exit	
Command	A	X	U	X	U
INITIALIZE	0	—	—	—	—
FORWARD	1	steps*	steps**	—	—
BACK	2	steps*	steps**	—	—
RIGHT	3	degrees*	degrees**	—	—
LEFT	4	degrees*	degrees**	—	—
Set X coordinate	5	X coord*	X coord**	—	—
Set Y coordinate	6	Y coord*	Y coord**	—	—
Set Heading	7	degrees*	degrees**	—	—
Set Pen Pos.	8	0 = up 1 = down	—	—	—
Set Turtle State	9	0 = hidden 1 = shows	—	—	—
Set Background	10	color code	—	—	—
Set Pen Color	11	color code	—	—	—
Clean Screen	12	—	—	—	—
Dot	13	X coord	Y coord	—	—
Set Text Pointer to Turtle position	14	—	—	—	—
Put Character	15	character	—	—	—
Read X coordinate	16	—	—	X coord*	X coord**
Read Y coordinate	17	—	—	Y coord*	Y coord**
Read Heading	18	—	—	degrees*	degrees**
Read Pen Position	19	—	—	0 = up 1 = down	0
Read Turtle State	20	—	—	0 = hidden 1 = shown	0
Read Background	21	—	—	color code	0
Read Pen Color	22	—	—	color code	0

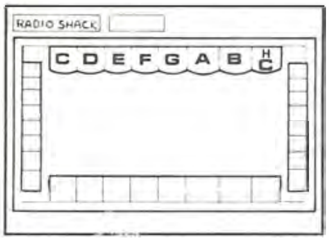
* – Integer portion only
** – Fractional portion times 65536 (e.g.: 0.5 = 32768)

Chapter Summary

PRIMITIVE	Abbrev.	Purpose
JOYX	–	Reads the X coordinate position of a specified joystick.
JOYY	–	Reads the Y coordinate position of a specified joystick.
BUTTON?	–	Determines if you press the specified joystick button.
PADX	–	Reads the X coordinate position of the X-Pad pen.
PADY	–	Reads the Y coordinate position of the X-Pad pen.
PADPENDOWN?	–	Determines if you press down the X-Pad pen.

Turtle Facts

- D.L. LOGO uses numbers to represent the 2 joysticks. The right joystick is number 0, and the left joystick is number 1.
- The joysticks produce X and Y coordinates in the range −32 to 31.
- The X-Pad produces X coordinates in the range −128 to 127 and Y coordinates in the range −96 to 95.



- To use remote devices with D.L. LOGO you must write a machine language module for that device.

Suggested Project

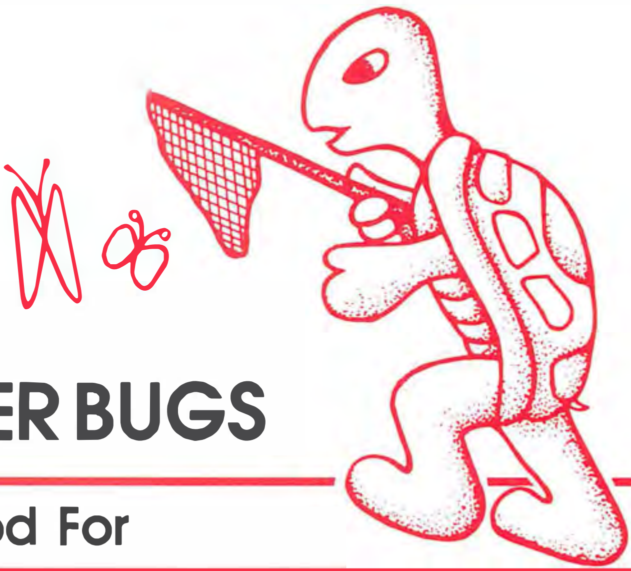
Draw a musical keyboard on a piece of paper on your X-Pad, and then write a program to play corresponding notes when you touch a key with the X-Pad pen.

Suggested Solution

```
TO PLAY
     LABEL "P
     WHILE NOT PADPENDOWN? []
     IF PADY>60
          [MUSIC LIST "T360 ITEM
            PADX/32+5 [C D E F G A B
            HC]
     WHILE PADPENDOWN? []
     GO "P
END
```

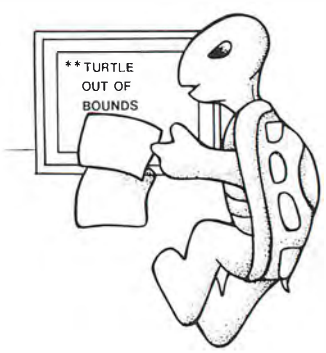
13 CHASING AFTER BUGS

What Errors Are Good For



Section 1	To Err Is Human: what are errors and how to avoid them.
Section 2	Debug: what to do when an error occurs.
Section 3	The Catch: how to catch errors and tame them.
Section 4	Error Code Reference.

Section 1
To Err Is Human

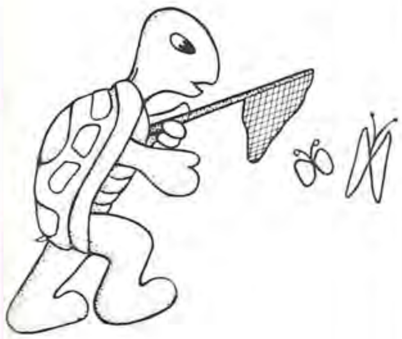


Errors are the bane of every programmer. Everyone makes mistakes when writing computer programs. Often, tracing and correcting these mistakes, or *bugs,* takes longer than writing the program.

This chapter deals with errors caused by faulty programming and errors caused purposefully. (Yes, errors can be useful in programming.) To aid in finding and correcting unwanted errors, D.L. LOGO's descriptive error messages tell you exactly what caused your problem. D.L. LOGO also shows you where an error occurs. Your task is to rewrite that part of the procedure so it can do its job.

Whenever an error occurs, LOGO stops the execution of the current procedure and displays an error message on the screen. When a procedure halts because of an error, you must find and correct the error before you can proceed. In short procedures, errors are much easier to find and correct than in long, complicated procedures. This is an important reason to keep your procedures short and simple.

When LOGO stops a procedure because of an error, it positions the editor where the error has occurred in the procedure. To see the line and position of the error, enter the edit mode. However, if you need to check the value of your variables, you must do so before entering the edit mode. Once in the edit mode, all variables are reset. The means for checking variables is discussed in more detail later in this chapter.



Section 2 Debug

Some errors are easy to spot and correct. For instance, if you make a typing error while entering the name of a primitive, LOGO stops the execution of the procedure and displays an error message such as:

```
** ERROR: UNDEFINED PROCEDURE
```

This means LOGO does not recognize your entry as a primitive name or a procedure name. To enter the edit mode and correct this error, type **EDIT** ENTER. The cursor is positioned at the location of the error. Suppose your procedure includes the primitive FORWARD misspelled as FOWARD. The line might look like this (the underline symbol represents the blue cursor):

```
RT 90 F_WARD 50
```

You only need to insert the letter R in FOWARD to correct the error; then reexecute the procedure.

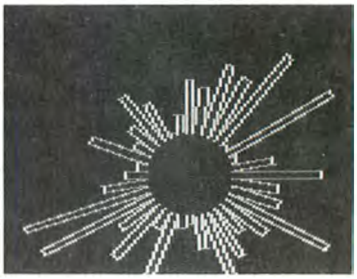
If you exit the edit mode without closing an opening bracket, LOGO immediately displays this error message:

```
** ERROR: MISSING "]"
```

In some cases, the cause of an error might not be so obvious. To demonstrate this, look at the following program. When you execute the procedure RING, an error is likely to halt the graphics display.

```
TO SPOKE
    WINDOW
    MAKE "M RANDOM 100
    MAKE "L RANDOM 3
    MAKE "R :M / :L
    FD :R RT 90
    FD 5 RT 90
```

```
          FD :R RT 190
          END

    TO RING
          CS
          SETY -10
          FULLSCREEN
          PU FD 20 PD
          REPEAT 36 [SPOKE]
    END
```



Enter the program and execute it by typing:

```
    ? RING
```

The screen displays a circle of random-length, radiating rectangles. Before the circle is complete, LOGO displays the message:

```
    ** ERROR: DIVIDE BY ZERO
```

When you enter the edit mode, the cursor is positioned on the variable L to locate the error. The cause of the error is in Line 3 of SPOKE. RANDOM 3 sometimes returns a 0, that gives L the value 0. You cannot divide a number by 0, as Line 4 is doing, and LOGO displays an error message.

There is a debugging aid that gives you a clue to this problem immediately. Execute the previous example again. When it stops with the divide by 0 error, type:

```
    ? CONTENTS [ENTER]
```

The screen looks similar to this:

```
    ? CONTENTS
    R=22
    L=0
    M=45
```

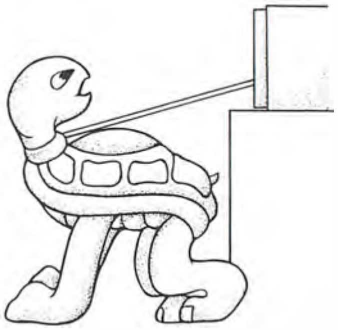
This is a list of all the variables in the program and their values. Since the variable M (45) is divided by the variable L (Ø), the cause of the error is easy to detect.

You can correct the problem by changing Line 4 in this fashion:

```
    MAKE "R :M / (:L+1)
```

Now L can never contain a value less than 1. The last section in this chapter describes all D.L. LOGO's error messages.

Section 3
The Catch

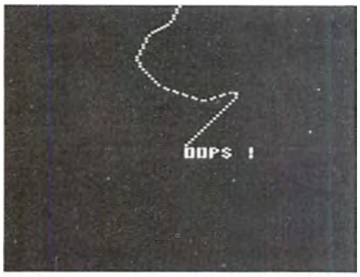


Sometimes D.L. LOGO's error-handling capabilities are helpful for programming. For instance, the FENCE primitive causes an error if the Turtle attempts to go beyond the video screen boundaries.

Suppose you are writing a program that requires the user to direct the Turtle around the screen, using a joystick. If you wish the turtle to stop at the edge of the video screen without creating an error message, use the primitive CATCH to intercept the error. To do this, first limit the Turtle to the screen boundaries using the FENCE primitive. If the Turtle tries to go beyond the screen boundaries, the message ''ERROR: TURTLE OUT OF BOUNDS'' is generated. Look up the error code reference at the back of this chapter; you see that the error number is 21.

The following procedure demonstrates a way to use CATCH.

```
TO JOY
     CS
     FENCE
     HT
     SETPC 3
     MAKE "X 0
     MAKE "Y 0
     AHEAD
END

TO AHEAD
     CATCH 21 [OOPS]
     WHILE NOT BUTTON? 0 [
          MAKE "X INT :X +(JOYX 0)/
            15
          MAKE "Y INT :Y +(JOYY 0)/
            15
```



```
                 DOT :X :Y]
    END

    TO OOPS
         TURTLETEXT [OOPS!]
         MAKE "X 0 MAKE "Y 0
         WAIT 20
         CS
         AHEAD
    END
```

Whenever the Turtle attempts to go out of bounds, CATCH causes program execution to go to the OOPS procedure. Instead of displaying the error TURTLE OUT OF BOUNDS, the word "OOPS" appears in the middle of the screen, the screen pauses, and the joystick positions reset to the center of the screen.

You can intercept any LOGO errors to provide an alternate error message, to reset values or conditions, or to direct program execution to another point. You cannot, however, return execution to the point where the error occurred.

Time Out

Using the PAUSE primitive is another way to examine program conditions. Insert PAUSE at any point in an errant procedure to halt execution while you examine current procedure conditions or change variable values. Then use CONTINUE to restart the procedure immediately following PAUSE. The following example uses PAUSE in conjunction with the CATCH primitive to stop the Turtle whenever it attempts to go beyond the screen boundaries.

```
    TO JOY
         CS
         FENCE
         HT
         SETPC 3
```

```
        MAKE "X 0
        MAKE "Y 0
        AHEAD
END

TO AHEAD
        CATCH 21 [PAWS]
        WHILE NOT BUTTON? 0 [
                MAKE "X INT :X +(JOYX 0)/
                  15
                MAKE "Y INT :Y +(JOYY 0)/
                  15
                DOT :X :Y]
END

TO PAWS
        PAUSE
        AHEAD
END
```

Now, when the Turtle goes out of bounds, the program halts and the cursor returns on the screen. When this happens, type **CONTENTS**. The screen shows the value of all variables:

```
Y=64
X=128
```

Now type:

```
? MAKE "X 0 MAKE "Y 0
? CONTINUE
```

The line of dots starts again at the center .he screen. PAUSE can be useful in debugging difficult programs. It lets you examine the conditions of a program, make changes to values if you wish, and continue execution at the point where the program stopped (or at some other point.) If you find that a program continuously crashes at a certain place, insert a CATCH-PAUSE routine to examine what is happening.

*. . . . **About Common Errors***

Although there are a number of errors D.L. LOGO recognizes, several are much more common than others. If you have trouble determining the cause of an error message, check these possibilities first:

- *Did you use dots instead of quotation marks in creating a variable?*
- *Did you use quotation marks to indicate a variable name instead of dots?*
- *Did you type a procedure name incorrectly?*
- *In the edit mode, did a line end at the extreme right of the screen, causing you to forget to press* ENTER *to complete the line?*
- *Did you forget to define a variable, using MAKE, before you used it as a value in a procedure?*
- *Do you have the same number of opening and closing parenthesis and brackets in a procedure?*

Some other uses of the CATCH primitive are:

- Catching the error if too few arguments are provided for a procedure or primitive. The operator could be asked to input more argument values.
- Informing you when your diskette is full while using a filing program.
- Telling you if a file cannot be found in the current directory.
- Directing a procedure to an alternate operation if an object becomes empty.
- Telling you if a number you are entering is out of range for its function.

Creating Errors

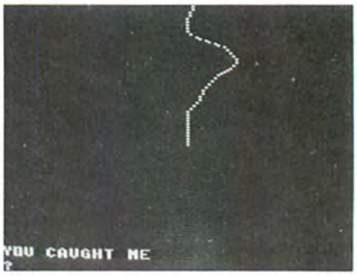
Not only does D.L. LOGO help you catch inadvertent errors, it also lets you simulate error conditions. Use the ERROR primitive to test *error traps* you created in your programs. To do so, set up an error trapping routine, then insert an ERROR command. For instance, you can use ERROR in the JOY program to test the CATCH 21 routine, without waiting until the Turtle reaches the edge of the screen. Add this line to the program:

```
TO AHEAD
    CATCH 21 [PAWS]
    ERROR 21
    WHILE NOT (BUTTON? 0) [
        MAKE "X INT :X+(JOYX 0)/15
        MAKE "X INT :Y+(JOYY 0)/15
        DOT :X :Y]]
END
```

```
   TO PAWS
         PRINT [YOU CAUGHT ME]
         PAUSE
         AHEAD
   END
```

A simulated ERROR 21 is created as soon as the TURTLE begins to move, and the program executes the PAWS procedure. Such a process can save considerable testing time.

ERROR can be used either from within a procedure or from the immediate, or single command, mode.



```
YOU CAUGHT ME
?
```

Section 4
Trace - How To Follow a Procedure

Sometimes errors seem impossible to find. When discovered, they are often simple mistakes, and you wonder how you missed such an obvious bug. At other times, an error is devious, and only persistent digging can uncover it.

The solution is knowing what a program is doing at all times. The TRACE primitive provides this information. Use the SPOKE and RING procedures in Section 2 to create the divide by zero error again. Before executing the program, enter the immediate mode, and type **TRACE 1Ø** ENTER. Now execute the program by typing **RING** ENTER.

This time, as the program runs, the screen displays each step of the procedure, 1Ø steps at a time. After each set of 1Ø steps is executed, press the space bar to continue the procedure. When :L is set to Ø in Line 2, the error in Line 4 causes the program to halt. At this point, the TRACE display resembles the following:

```
M
RANDOM
1ØØ
35
MAKE
L
RANDOM
3
Ø
MAKE
R
35
 Ø
** ERROR: DIVIDE BY ZERO
```

Each step of the procedure and each variable are displayed as the program runs. By looking at the TRACE list you see that the value of L is 0 and the value of R is 35. LOGO cannot divide by 0; thus, the error occurs. To turn the TRACE function off, type **NOTRACE** ENTER. You can set the step rate of the TRACE function to a maximum value of 255. If no argument is used, TRACE does not cause the program to pause during execution.

If you have a printer, you can use the COPYON primitive to print the entire TRACE list. Doing so provides a hard copy of the entire program execution to the point of the error. Use COPYOFF to stop screen output from echoing to your printer.

Section 5
Error Code Reference

CODE #	MESSAGE	DESCRIPTION
1	MUST BE A WORD	A *list* was supplied as an argument to a procedure requiring a *word*.
2	MUST BE A NUMBER	An object other than a number was supplied as an argument to a procedure requiring a number.
3	NOT TRUE/ FALSE	A primitive, designed to determine TRUE/FALSE conditions, was supplied in an argument that could not be established as either TRUE or FALSE.
4	MUST BE A LIST	A *word* was supplied as an argument to a procedure that requires a *list*.
5	EMPTY OBJECT	An empty object was passed to a FIRST BUTFIRST LAST or BUTLAST primitive.
6	NUMBER OUT OF RANGE	The number supplied to an operation was too low or too high for the capabilities of that operation.
7	MISSING ARGUMENT	A procedure or function was supplied fewer arguments than it requires.
8	DIVIDE BY ZERO	An attempt was made to divide a number by Ø.

CODE #	MESSAGE	DESCRIPTION
9	UNDEFINED SYMBOL	The label referenced by THING or dots (:) was not defined.
10	ISOLATED OBJECT	An attempt was made to use an argument (or value) without an associated procedure.
11	MUST BE A PROC LIST	A *word* was supplied as an argument to an operation that requires a procedure *list*.
12	SYNTAX ERROR	Invalid syntax was used in a procedure.
13	UNDEFINED PROCEDURE	A name was used for which no procedure exists.
14	BAD 'TO' STATEMENT	A syntax error exists in Line 1 of a procedure.
15	MEMORY FULL	Ram space allocated to LOGO has been exhausted.
16	MISSING ']'	Either too many or too few brackets are included in a procedure.
17	MISSING ")"	Either too many or too few parentheses are included in a procedure.
18	BAD PROCEDURE CALL	A LOGO primitive procedure is being called improperly. For example, EDIT is being called from a user program.
19	NOT FOUND	The specified pathname was not found.

CODE #	MESSAGE	DESCRIPTION
20	DISK ERROR	An OS-9 disk error has occurred.
21	TURTLE OUT OF BOUNDS	The Turtle attempted to go beyond the screen boundaries while the FENCE primitive was active.
22	CAN'T ERASE FILE	A disk error occurred while OS-9 was attempting to delete a file.
23	CAN'T OPEN FILE	OS-9 has returned an error while attempting to open a file.
24	FILE NOT OPEN	A file read or write operation was called before an OPENREAD or OPENWRITE was implemented.
25	DISK FULL	Diskette space has been exhausted.
26	WORD TOO LONG	A word was used that exceeds 255 characters.
27	PROCEDURE ALREADY DEFINED	A procedure name was used that was either a LOGO primitive name or that was already defined as a procedure.
28	SQRT OF NEG NUMBER	An attempt was made to calculate the square root of a negative number.
29	LOG OF NEG NUMBER	An attempt was made to calculate the LOG of a negative number.

CODE #	MESSAGE	DESCRIPTION
30-32	User definable	Machine-language programmers can use these codes to create error messages for an external device. For more information on controlling external devices, see Chapter 11.

Chapter Summary

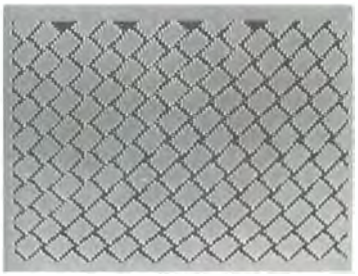
PRIMITIVE	Abbrev.	Purpose
CONTENTS	–	Lists all global variables.
FENCE	–	Causes an error if the Turtle goes beyond the video screen boundaries.
CATCH	–	Redirects LOGO's normal error function.
ERROR	–	Simulates an error condition.
TRACE	–	Turns on LOGO's trace function, displays procedure steps.
NOTRACE	–	Turns off LOGO's trace function.

Turtle Facts

- Unless CATCH is specified, a procedure error causes a procedure to halt.
- The CATCH primitive lets you create your own error messages or select a routine other than normal error execution.
- LOGO's editor cursor marks the location in a procedure where an error occurs.
- Variable values are lost when you enter the edit mode.
- You can create a permanent trace of a program's execution by using both the COPYON and TRACE primitives.

Proposed Project

Write a program to draw bricks diagonally across the screen. Use the CATCH primitive to: (1) determine when a row of bricks is completed and (2) cause the procedure to begin the next row. Finish the project with a border of your choice by using the CATCH primitive to: (1) determine when the border has reached the edge of the screen and (2) cause the procedure to begin the next side of the border.

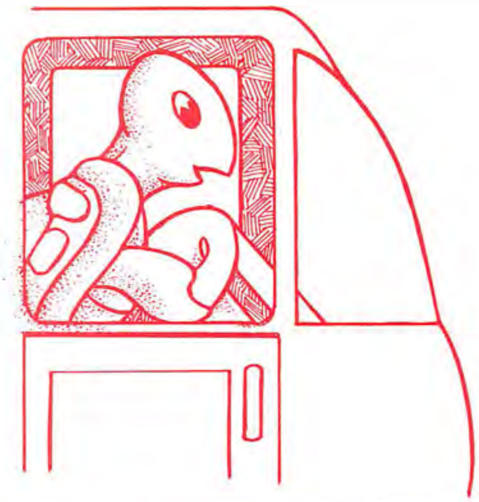


Suggested Solution

```
TO BRICKS
     HT
     CATCH 21 [SET]
     FENCE
     FULLSCREEN
     CS
     MAKE "Y 86
     MAKE "X -128
     ROW
END

TO SET
     IF :X>120 [BORDER]
     IF :Y>-80 [MAKE "Y :Y-19] ELSE
      [MAKE "X :X+22]
     ROW
END

TO ROW
     SETXY :X :Y
     SETH 45
     REPEAT 100 [BRICK]
END

TO BRICK
     SETPC 0 FD 15
     SETPC 2 BK 15
     REPEAT 6 [
          FD 15
          RT 90
          FD 1
          RT 90
          FD 15
          LT 90
          FD 1
          LT 90]
     PU LT 90 FD 11 RT 90 FD 19 PD
END
```

```
TO BORDER
     SETPC 2
     SETXY -128 96
     SETH 0
     MAKE "X 0
     BAND
END

TO BAND
     FENCE
     CATCH 21 [NEXT]
     REPEAT 257 [
          RT 180
          FD 12
          LT 90
          FD 1
          LT 90
          FD 12]
END

TO NEXT
     WINDOW
     LT 90
     FD 12
     RT 90
     BK 1
     MAKE "X :X+1
     IF :X=4 [TOPLEVEL]
     BAND
END
```

14 TURTLE IN THE DRIVER'S SEAT



Taking Charge of Files, OS-9, and Screen Format

Section 1	Breaking Out: introducing the capabilities of D.L. LOGO's file-handling procedures.
Section 2	Fine Tuning File Control: advanced methods of accessing files.
Section 3	Calling on OS-9: using OS-9 commands, entering the OS-9 SHELL from LOGO, and returning to LOGO.
Section 4	Turtle on the Textscreen: arranging screen displays.

Section 1 Breaking Out

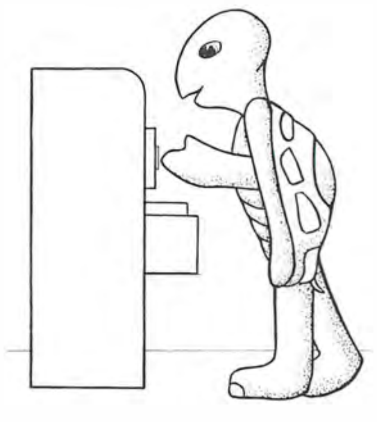
The LOGO language is well known for its graphics capabilities, but D.L. LOGO also has sophisticated text handling features. This chapter explains how to manipulate data files on your disk drive, how to access OS-9 commands, and how to format the text screen in the immediate mode.

D.L. LOGO uses the OPENWRITE primitive to open a file for storing data. If the file you name with OPENWRITE does not exist, LOGO creates it. If it does exist, LOGO opens it and appends new data to the previously stored data. For instance, to open a file named *BOOK*, type **OPENWRITE "BOOK**. You can also use the OPENWRITE primitive with variable names. If the variable NAME contains the value BOOK, the command OPENFILE :NAME produces exactly the same result as OPENFILE "BOOK.

The following procedure asks you to enter a name for a file you want to open. If the file already exists, the procedure opens it to receive further input. If the file does not exist, the OPENWRITE primitive creates a file with the name you specify and opens it for input. The variable NAME stores the name you give the file.

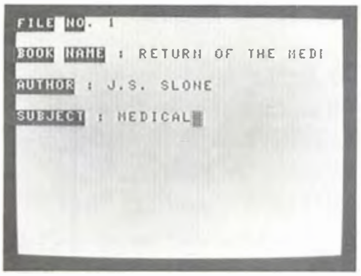
The FIRST primitive used in Line 5 causes the variable NAME to be a word rather than a list. The OPENWRITE primitive accepts only words as filenames.

```
    TO CREATE
         CLEARTEXT
         SETCURSOR 5 0
         PRINT1 [NAME OF FILE:]
         MAKE "NAME FIRST RQ
         OPENWRITE :NAME
         INPUT
    END
```



. . . . ***About Files***

The term files *indicates any data saved on a diskette. This can include procedures and programs as well as data from a procedure or program. For instance, you can write a program to keep a record of your personal library. When you complete the program, save it on diskette so that it is available to index your books. You might name the file* BOOKS. *When you run the program, it creates a disk file that contains information about your books. You can name the file* BOOKINDEX. *Although the 2 files contain different types of data, to LOGO they are both files. You can load both data and procedure files into LOGO's workspace.*



```
FILE NO. 1

BOOK NAME : RETURN OF THE MEDI

AUTHOR : J.S. SLONE

SUBJECT : MEDICAL
```

. . . . *About INPUT*

This program prompts you to enter a bookname, the author, and the subject. Type the information you wish to use, then press ENTER *after each input. When you have entered as many files as you wish, type END* ENTER *in response to the bookname prompt. This will cause the procedure to end execution and exit to the immediate mode. Be sure you type the following COMBINE procedure before attempting to create files with INPUT.*

The preceding procedure opens a file, but you use another procedure to store something in it. The last line of the CREATE procedure calls the following INPUT procedure, which creates a file of the books in your library. It asks for the names, authors, and subjects of your books:

```
TO INPUT
     CLEARTEXT
     PRINT
     PRINT [book name:]
     PRINT
     PRINT [author:]
     PRINT
     PRINT [subject:]
     SETCURSOR 1 12
     MAKE "BOOKNAME RQ
     IF :BOOKNAME=[END]
      [CLOSEWRITE :NAME TOPLEVEL]
     SETCURSOR 3 9
     MAKE "AUTHOR RQ
     SETCURSOR 5 10
     MAKE "SUBJECT RQ
     COMBINE
END
```

This procedure displays 3 fields on the screen: *BOOKNAME*, *AUTHOR*, and *SUBJECT*. It places the cursor at the end of each field name and waits for you to enter data.

As yet, the program does not save anything on diskette. The last line of the INPUT procedure, calls the COMBINE procedure to do this. COMBINE combines the 3 entries into 1 record and saves it in the newly created file with the WRITE procedure. After the procedure has written an entry to diskette, it returns to the INPUT procedure for another entry.

```
TO COMBINE
     MAKE "BOOK LIST :BOOKNAME
       :AUTHOR :SUBJECT
```

```
        WRITE :NAME :BOOK
        INPUT
    END
```

Now look back at Line 11 of the INPUT procedure. This line makes it possible for you to exit the program when you finish entering data by typing the word "END" as the book name. When you do so, the line closes the open file and causes the program to end.

To record the number of entries you make, add the following as Line 2 in the CREATE procedure:

```
    MAKE "COUNT 0
```

Before the last line of the INPUT procedure, insert:

```
    MAKE "COUNT :COUNT+1
```

To remind you of the current file as you add data, insert this line as Line 3 of the INPUT procedure:

```
    PRINT [file no. ] :COUNT+1
```

Then add 1 to the present cursor position in the INPUT procedure by changing the following lines as shown:

```
    Line 10: SETCURSOR 2 12
    LINE 14: SETCURSOR 4 9
    Line 16: SETCURSOR 6 10
```

Reading the Writing on Diskettes

Reading files into D.L. LOGO from diskette is as easy as writing them. The primitives you use are OPENREAD, READ and CLOSEREAD. For instance, type:

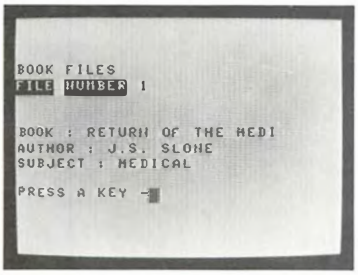
```
    ? OPENREAD "BOOKFILE
    ? MAKE "BOOK READ "BOOKFILE
    ? CLOSEREAD "BOOKFILE
```

. . . . *About Records*

The term records *refers to an entry into a file. In this section, the WRITE primitive combines the fields "BOOKNAME," "AUTHOR," and "SUBJECT" into 1 record, and places it in a file.*

. . . . About Fields

The term field *refers to a keyboard entry. In the examples used in this section, "BOOKNAME," "AUTHOR," and "SUBJECT," are fields.*



```
BOOK FILES
FILE NUMBER 1

BOOK : RETURN OF THE MEDI
AUTHOR : J.S. SLONE
SUBJECT : MEDICAL

PRESS A KEY -
```

These lines open a file named BOOKFILE, read the first entry, and close the file. To read all the entries in a file, you use a procedure that places the *read entry* process in a loop and determines when it reaches the end of the file. If you use a variable to store data from a file, you can use EMPTY? to determine when you reach the end of a file. For instance, if the variable BOOK stores the elements of a file, the command PRINT EMPTY? :BOOK produces a value of TRUE when you reach the end of that file.

The following procedure uses this method as it opens and reads the files saved by the preceding INPUT procedure. It reads a file, 1 record at a time, divides the record into its 3 fields, formats the screen, and displays the fields. When the procedure reaches the end of the file, BOOK becomes empty and the screen displays the message, "END OF FILES":

```
TO SHOWFILE
     PRINT1 [NAME OF FILE...]
     MAKE "NAME FIRST RQ
     OPENREAD :NAME
     LABEL "FILE
     MAKE "BOOK READ :NAME
     IF NOT EMPTY? :BOOK [
          MAKE "BOOKNAME FIRST :BOOK
          MAKE "AUTHOR ITEM 2 :BOOK
          MAKE "SUBJECT LAST :BOOK
          CLEARTEXT
          SETCURSOR 5 5
          PRINT [FILES FOR] :NAME
          SETCURSOR 7 0
          PRINT [BOOK:] :BOOKNAME
          PRINT [AUTHOR:] :AUTHOR
          PRINT [SUBJECT:] :SUBJECT
          SETCURSOR 11 0
          PRINT [PRESS A KEY-]
          MAKE "NUL RC
          GO "FILE]
     PRINT [END OF FILES...]
     CLOSEREAD :NAME
END
```

Although this procedure seems long, most of it deals with formatting the screen. The commands that access the file appear in Lines 2 and 3 and in the last line. Line 2 opens the file for reading, Line 3 reads an item from the file, and the last line closes the read file. In this procedure, IF NOT EMPTY? :BOOK determines when the procedure reaches the end of the file.

Section 2 Fine Tuning File Control

The primitives WRITEBYTE, READBYTE, FILEPOS, and SETFILEPOS give you additional control over your filing needs. The WRITEBYTE and READBYTE primitives let you write to and read from files, 1 byte rather than 1 item at a time. The FILEPOS and SETFILEPOS primitives let you determine a file's current position and set new positions for reading and writing.

Using FILEPOS and SETFILEPOS, you can read any portion of a file without reading through all previous file entries. Using the WRITEBYTE and READBYTE primitives, you can directly read or change any character in a record or file.

The following examples show you how to use these primitives using a sample filename TEST:

Action	Command
Open a file to store data	OPENWRITE "TEST
Write to the file	WRITE "TEST [HELLO]
Close the file	CLOSEWRITE "TEST
Open a file to read data	OPENREAD "TEST
Read data and store in variable DATA	MAKE "DATA READ "TEST
Close the file	CLOSEREAD "TEST
Open a file to write a byte	OPENWRITE "TEST
Set the file position to write a byte	SETFILEPOS "TEST 3
Write a new byte at file position 3	WRITEBYTE "TEST "65
Close the file	CLOSEWRITE "TEST

Action	Command
Open a file to read a byte	OPENREAD "TEST
Set the file position to read a byte	SETFILEPOS "TEST 2
Read a byte and store in variable DATA	MAKE "DATA READBYTE "TEST
Close the file	CLOSEREAD "TEST

Not only are these file handling capabilities easy to understand and use, but they also open data processing capabilities that few other systems offer. For instance, you can create an index to keep track of the file position of each record as you write on the diskette. Using this index, you can create procedures to sort records and rearrange the the index, rather than rearranging the records on a diskette. You can also use the index to provide direct access to any record, regardless of its position in a file. The following program shows how to do this.

```
TO ADD
        MAKE "INDEX [0]
        CLEARTEXT
        OPENWRITE "TEST
        FOR "I 1 9 1 [
             CLEARTEXT
             SETCURSOR 0 14
             PRINT [ADD]
             MAKE "DATA RQ
             WRITE "TEST :DATA
             MAKE "INDEX LPUT FILEPOS
             "TEST :INDEX]
        CLOSEWRITE "TEST
        OPENWRITE "TESTX
        WRITE "TESTX :INDEX
        CLOSEWRITE "TESTX
        SEE
   END
```

```
TO SEE
     OPENREAD "TESTX
     MAKE "INDEX READ "TESTX
     CLOSEREAD "TESTX
     OPENREAD "TEST
     LABEL "NEXT
     PRINT1 [ENTRY NUMBER?...]
     MAKE "ENTRY RC
     SETFILEPOS "TEST ITEM :ENTRY
     :INDEX
     MAKE "RECORD READ "TEST
     PRINT :RECORD
     GO "NEXT
END
```

This program writes data (whatever you type) into a disk file called TEST. At the same time it keeps an index of where each item is stored in the file. This information is kept in the variable INDEX. When you have typed 10 entries, the index is stored in another disk file, called TESTX.

The SEE procedure, reads this index back into LOGO's memory and reopens the TEST file. You can press any key in the range of 1-9 to see the corresponding entry. When you press a number key, the location of the indicated record is found in the INDEX variable, and the disk file pointer is set to that location. The indicated record in the file is then read and displayed.

For instance, if you type the following 9 entries:

```
ONE ENTER
TWO ENTER
THREE ENTER
FOUR ENTER
FIVE ENTER
SIX ENTER
SEVEN ENTER
EIGHT ENTER
NINE ENTER
```

You can type **6** from the SEE procedure, and the screen displays SIX. If you type **2**, the screen displays TWO.

Get a Bit With BYTE

The READBYTE and WRITEBYTE primitives let you access single characters in a file. This feature provides a number of options for data file management. For instance, you can create a data file that combines into 1 file both data records and an index to manage those records. To create a file in which to experiment, type the following:

```
? OPENWRITE "TEST ENTER
? WRITE "TEST "0123456789 ENTER
? CLOSEWRITE "TEST ENTER
```

Now, to see if things are where they should be, type:

```
? OPENREAD "TEST ENTER
? MAKE "A READBYTE "TEST ENTER
? CLOSEREAD "TEST ENTER
? PRINT :A ENTER
48
```

Because you haven't told it otherwise, READBYTE reads the first byte of the TEST file and displays a value of 48. But 48 isn't what you stored at byte 0. Byte 0 should contain 0. Perhaps it does. Try typing **PRINT CHAR :A** to produce an accurate result. The variable A shows a value of 48 because D.L. LOGO stores the characters you send to a file as ASCII values (48 is the ASCII value of 0). READBYTE returns an ASCII value, and the CHAR primitive produces the original character.

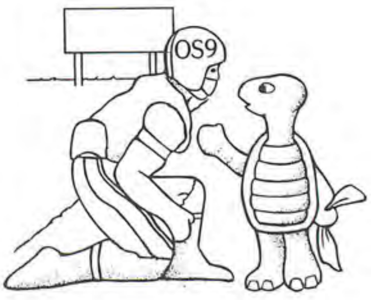
Issuing the READBYTE primitive automatically increases the counter indicating the file position. To see the second byte of the file, type **MAKE "A READBYTE "TEST** ENTER.

The variable A now contains the value of the second byte. Test this by typing **PRINT CHAR :A**. The screen displays the number 1.

Going On

This manual only touches on the techniques and procedures of file management. Determine your special needs and put D.L. LOGO to the test. You will find it ready to meet most challenges. D.L. LOGO is as capable of data processing as it is of producing superb graphics.

Section 3
Calling On OS-9



D.L. LOGO runs on the OS-9 disk operating system. If you have the OS-9 system diskette, all the system commands and utilities are available to you through the D.L. LOGO SHELL primitive. In OS-9, the SHELL is a command interpreter. Using the SHELL primitive transfers commands to the OS-9 SHELL program. You can copy OS-9 commands and utilities to your D.L. LOGO diskette, then use them from LOGO with the SHELL primitive. For instance, if you copy the DIR command to your D.L. LOGO diskette, you can use it to view a directory by typing:

```
? SHELL [DIR /D0]
```

See your OS-9 Commands manual for information on how to copy and use commands and utilities.

An Alternate Method

You can also employ the SHELL primitive to exit D.L. LOGO and use OS-9 commands directly. After you finish executing the OS-9 commands you wish to use, a 2-key process lets you reenter LOGO. To try this process, exit D.L. LOGO by typing the SHELL primitive with an empty procedure list:

```
SHELL [] ENTER
```

D.L. LOGO enters the OS-9 SHELL through the SHELL primitive. Now, you can use an OS-9 system diskette and execute any commands or utilities you wish. (See your OS-9 manuals for command and utility information.) To reenter D.L. LOGO, hold down CTRL and press BREAK. LOGO's question mark (?) prompt reappears, and you can again execute LOGO primitives and procedures.

Section 4
Turtle on the Text Screen

Although D.L. LOGO lets you display text in both the immediate and graphics modes, several primitives are associated with text displays that let you format and arrange the text as you wish.

The PRINT and PRINT1 primitives are used in demonstration programs throughout the manual. PRINT causes an automatic carriage return after the text it displays. PRINT1 does not cause an automatic carriage return. The following examples demonstrate how PRINT and PRINT1 operate when used in a procedure:

```
PRINT [HELLO]
PRINT [PAUL]
```

When the procedure executes these lines, the screen shows:

```
HELLO
PAUL
```

However, when you use the following lines in a procedure:

```
PRINT1 [HELLO]
PRINT [PAUL]
```

The screen shows:

```
HELLO PAUL
```

By Line and Column

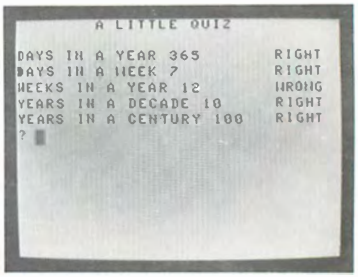
The COLUMN primitive tells you the current column position of the cursor (from 0 to 31), and the LINE primitive tells you the current line position (from 0 to 15). You can use the COLUMN and LINE primitives with the SETCURSOR primitive to reposition the text screen's cursor after a print. For instance, the following procedure displays a line of asterisks diagonally down the text screen:

```
TO DIAG
     SETCURSOR Ø Ø
     REPEAT 15 [
     PRINT1 [*]
     SETCURSOR LINE+1 COLUMN+1]
END
```

The line and column position increases in increments of 1 each time an asterisk is displayed. Because you use the PRINT1 primitive, each print location is dependent on the SETCURSOR primitive.

The following program creates a simple quiz, showing how the screen might be formatted:

```
TO QUIZ
     CLEARTEXT
     GETSET
     MAKE "B CHAR 32
     SETCURSOR Ø 8
     PRINT [A LITTLE QUIZ]
     SETCURSOR 2 Ø
     FOR "T 1 5 1[
          PRINT1 THING (WORD "Q :T)
           :B
          MAKE "A RQ
          CHECK
          SETCURSOR LINE+1 Ø]
END

TO CHECK
     IF :A=(THING WORD "A :T) [
          MAKE "RESPONSE "RIGHT]
          ELSE [MAKE "RESPONSE
           "WRONG]
```



```
          A LITTLE QUIZ

DAYS IN A YEAR 365            RIGHT
DAYS IN A WEEK 7              RIGHT
WEEKS IN A YEAR 12            WRONG
YEARS IN A DECADE 10          RIGHT
YEARS IN A CENTURY 100        RIGHT
?
```

```
          SETCURSOR LINE-1 25 PRINT1
           :RESPONSE
END

TO GETSET
          MAKE "Q1 [DAYS IN A YEAR?]
          MAKE "Q2 [DAYS IN A WEEK?]
          MAKE "Q3 [WEEKS IN A YEAR?]
          MAKE "Q4 [YEARS IN A DECADE?]
          MAKE "Q5 [YEARS IN A CENTURY?]
          MAKE "A1 [365]
          MAKE "A2 [7]
          MAKE "A3 [52]
          MAKE "A4 [10]
          MAKE "A5 [100]
END
```

Chapter Summary

PRIMITIVE	Abbrev.	Purpose
OPENWRITE	–	Opens a file for data input.
CLOSEWRITE	–	Closes a file you opened using OPENWRITE.
WRITE	–	Transfers data to a specified file.
OPENREAD	–	Opens a file for reading data.
CLOSEREAD	–	Closes a file that you opened using OPENREAD.
READ	–	Accesses data in a specified file.
FILEPOS	–	Provides the current position of the file pointer.
SETFILEPOS	–	Sets the file pointer to a specified position.
READBYTE	–	Reads 1 byte of file data at a file's current position. The file position pointer automatically increases.

PRIMITIVE	Abbrev.	Purpose
WRITEBYTE	–	Writes 1 byte of file data to the current file position. The file's position pointer automatically increases.
SHELL	–	Lets you to access OS-9 commands from within D.L. LOGO.
PRINT	–	Displays specified characters on the text screen and ends with an automatic carriage return.
PRINT1	–	Displays specified characters on the text screen, but does not cause a carriage return.
COLUMN	–	Returns the current column position of the text screen cursor.
LINE	–	Returns the current text line position.
SETCURSOR	–	Establishes the cursor at specified line and column position.

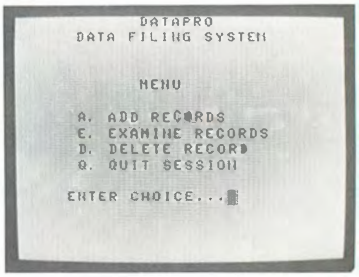
Turtle Facts

- D.L. LOGO has sophisticated file and data management capabilities.
- Direct access files are possible with D.L. LOGO.

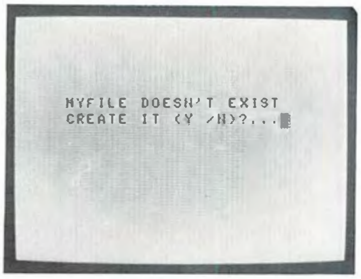
- Always use CLOSEWRITE or CLOSEREAD to close files when you finish writing or reading them.
- You can move D.L. LOGO's file position pointer at will.
- Data is stored as ASCII code.
- LOGO lets you access all OS-9 commands without leaving the LOGO program.
- You can exit from and return to LOGO without losing data.
- Screen format commands can also format printer output.

Suggested Project

Use the information and sample procedures in this chapter to write a useful file management program. The following sample program is written as a general purpose filing program with features to create files, add to existing files, and delete files. FILEPOS and SETFILEPOS are used to set a file pointer index that lets you directly access any file. These primitives also let you create records of any length, and with any number of fields. You could improve this program by using the index for file sorting or inserting routines.

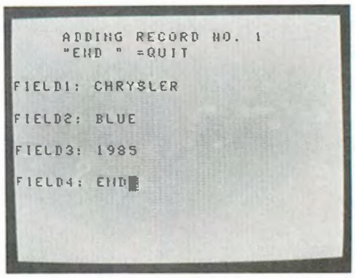


```
          DATAPRO
     DATA FILING SYSTEM

           MENU

     A. ADD RECORDS
     E. EXAMINE RECORDS
     D. DELETE RECORD
     Q. QUIT SESSION

    ENTER CHOICE...
```

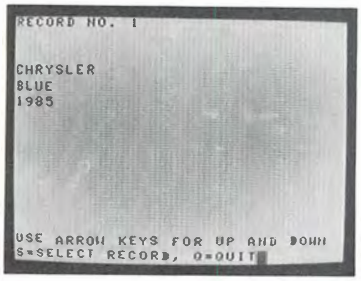

```
MYFILE DOESN'T EXIST
CREATE IT (Y /N)?...
```

```
;-------------------
; DATA FILING SYSTEM
;-------------------
TO DATA
     WHILE "TRUE [CLEARTEXT
          C 0 12
          PRINT [DATAPRO]
          C 1 6
          PRINT [DATA FILING SYSTEM]
          C 4 12 PRINT [m e n u]
          C 6 6
          PRINT [A. ADD RECORDS]
          C 7 6
          PRINT [E. EXAMINE RECORDS]
          C 8 6
          PRINT [D. DELETE RECORD]
          C 9 6
          PRINT [Q. QUIT SESSION]
          C 11 5
          PRINT1 [ENTER CHOICE...]
          MAKE "CHOICE RC PICK]
END

TO PICK
     SELECT [
          :CHOICE="A [ADD]
          :CHOICE="E [EXAM]
          :CHOICE="D [DELETE]
          :CHOICE="Q [QUIT]
          "TRUE [PUT [CHOICE INVALID]]]
END

TO ADD
     CLEARTEXT
     C 0 0
     PRINT [* * * * * ADD RECORDS * * *
      * *]
     IF GETFILE [INPUT] ELSE
     [CLEARTEXT
     C 5 5
```

```
      PRINT :FILE [DOESN'T EXIST]
      C 6 5
      PRINT1 [CREATE IT \(Y/N\)?...]
      IF (FIRST FIRST RQ)="Y
           [CREATE INPUT]]
END

TO INPUT
      OPENREAD :FILEX
      MAKE "INDEX SE READ :FILEX
      CLOSEREAD :FILEX
      OPENWRITE :FILE
      DO [CLEARTEXT
           C 1 5
           PRINT [ADDING RECORD NO.]
            COUNT :INDEX+1
           C 2 5
           PRINT ["END"=QUIT]
           MAKE "RECORD []
           MAKE "NO 0
           DO [MAKE "NO :NO+1
              PRINT
              PRINT1 "FIELD :NO [:\ ]
              MAKE "RECORD LPUT RQ
               :RECORD]
           WHILE (LAST :RECORD)<>[END]
      MAKE "INDEX LPUT FILEPOS :FILE
       :INDEX
      WRITE :FILE BUTLAST :RECORD
      CLEARTEXT
      C 0 5
      PRINT1 [ANOTHER RECORD \(Y/
       N\)?...]]
      WHILE (FIRST FIRST RQ)="Y
      CLOSEWRITE :FILE
      ERASEFILE :FILEX
      OPENWRITE :FILEX
      WRITE :FILEX :INDEX
      CLOSEWRITE :FILEX
END
```



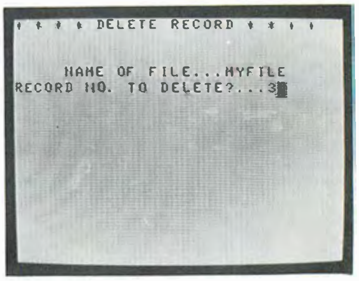
```
     ADDING RECORD NO. 1
     "END " =QUIT

FIELD1: CHRYSLER

FIELD2: BLUE

FIELD3: 1985

FIELD4: END
```



```
RECORD NO. 1


CHRYSLER
BLUE
1985






USE ARROW KEYS FOR UP AND DOWN
S=SELECT RECORD, Q=QUIT
```

```
TO CREATE
     OPENWRITE :FILEX
     WRITE :FILEX []
     CLOSEWRITE :FILEX
     OPENWRITE :FILE
     CLOSEWRITE :FILE
END

TO EXAM
     CLEARTEXT
     C 0 0
     PRINT [* * * * EXAMINE RECORDS * *
      * *]
     IF NOT GETFILE
          [PUT [FILE NOT FOUND]
          STOP]
     OPENREAD :FILEX
     MAKE "INDEX READ :FILEX
     CLOSEREAD :FILEX
     OPENREAD :FILE
     MAKE "R 1
     DO [
          IF :R<1 [MAKE "R 1]
          IF :R>(COUNT :INDEX) [MAKE "R
           COUNT :INDEX]
          SETFILEPOS :FILE ITEM :R
           :INDEX
          MAKE "RECORD READ :FILE
          DISPLAY
          C 14 0
          PRINT [USE ARROW KEYS FOR UP
           AND DOWN]
          PRINT1 [S\=SELECT RECORD,
           Q\=QUIT]
          DO [MAKE "CHOICE RC]
          WHILE NOT MEMBER? :CHOICE LIST
           CHAR 12 CHAR 10 "S "Q
          SELECT [
             :CHOICE=CHAR 12 [MAKE "R
              :R-1]
```

```
                 :CHOICE=CHAR 10 [MAKE "R
                  :R+1]
                 :CHOICE="S [SELCT]]
          WHILE :CHOICE <>"Q
END

TO SELCT
     C 14 0
     P 63
     PRINT [-SELECT RECORD-]
     PRINT1 [RECORD NUMBER\:\ ]
     MAKE "R FIRST RQ
END

TO DISPLAY
     CLEARTEXT
     PRINT [record no.] :R
     C 3 0
     FOR "X 1 COUNT :RECORD 1 [
          PRINT ITEM :X :RECORD]
END

TO DELETE
     CLEARTEXT
     PRINT [\ * * * * DELETE RECORD \* *
      * *]
     IF NOT GETFILE
          [PUT [FILE NOT FOUND]
          STOP]
     OPENREAD :FILEX
     MAKE "INDEX READ :FILEX
     CLOSEREAD :FILEX
     PRINT1 [RECORD NO. TO DELETE?...]
     MAKE "D FIRST RQ
     MAKE "END COUNT :INDEX
     SELECT [
          :D=1 [MAKE "INDEX BUTFIRST
           :INDEX]
          :D=:END [MAKE "INDEX BUTLAST
           :INDEX]
```

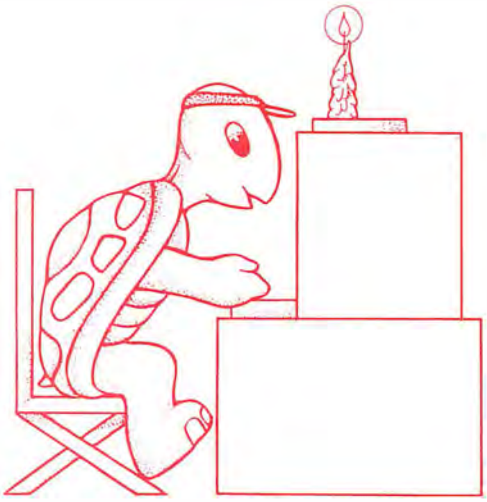


```
* * * * DELETE RECORD * * * *

    NAME OF FILE...MYFILE
RECORD NO. TO DELETE?...3
```

```
          ALLOF :D>1 :D<:END
             [MAKE "INDEX SE PIECE 1 :D-
              1 :INDEX PIECE :D+1 :END
              :INDEX]]
     ERASEFILE :FILEX
     OPENWRITE :FILEX
     WRITE :FILEX :INDEX
     MAKE "N COUNT :INDEX
     C 11 0
     PUT SE :FILE "NOW "HAS :N "RECORDS
END

TO QUIT
     CLEARTEXT
     C 8 5
     PRINT [session ended by user]
     TOPLEVEL
END

TO GETFILE
     C 3 5
     PRINT1 [NAME OF FILE...]
     MAKE "FILE FIRST RQ
     MAKE "FILEX WORD :FILE "X
     OUTPUT MEMBER? :FILE CATALOG
END

TO PUT :MES
     C 14 0
     P 63
     PRINT :MES
     PRINT1 [\(PRESS ANY KEY TO
      CONTINUE...\)]
     MAKE "SLUFF RC
     C 14 0
     P 63
END
```

```
TO P :N
     MAKE "L LINE
     MAKE "C COLUMN
     PRINT1 PIECE 1 :N "\ \ \ \ \ \ \ \
      \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
      \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
      \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
      \ \ \ \ \ \
     C :L :C
END
```

15 EDITING: THE FULL STORY



Advanced Editing Techniques

Section 1	Editing Features: some of D.L. LOGO's advanced editing features and how to use them.
Section 2	The Search: searching for and replacing text.
Section 3	Edit Commands Reference: a summary of all editing commands and key functions.

Section 1 Editing Features

In Chapter 3 you learned D.L. LOGO's basic editing functions. Many other features make program writing and editing much easier and quicker. Some of these features include the ability to:

- page forward or backward through text
- move the cursor forward and backward one word at a time
- move the cursor to the beginning or end of a line or of the entire workspace text
- delete specified numbers of words or characters
- delete to a specified character
- delete blocks of text
- change a character, word, or line
- move and duplicate blocks of text
- repeat edit commands
- *unedit* changed text
- search text for a specified word, value, or character
- search and change one or all occurrences of a specified word, value, or character

It is worth your time to learn about these commands and, as you write programs and procedure, to refer to this chapter often. This section describes many advanced editing commands available. A chart at the end of the chapter provides a quick reference to all the editing commands.

Inserting in Text

From the edit mode, you can enter the insert mode in several ways. Pressing I begins an insert at the cursor position. You continue in the insert mode until you press BREAK. Pressing SHIFT I positions the cursor for insert at the beginning of the line in which the cursor is located.

Pressing A begins an insert immediately following the cursor position. SHIFT A begins an insert at the end of the line in which the cursor is located.

To begin an insert immediately after the the line in which the cursor is located, press O. To begin an insert immediately before the line in which the cursor is located, press SHIFT O.

Deleting Text

The delete commands lets you remove characters, words, lines, or blocks of text from your workspace. To delete a character, position the cursor over that character and press X. You can continue pressing X to delete as many characters as you wish. Typing ***n***X deletes the next *n* characters (*n* is the number of characters you wish to delete). To delete the character to the left of the cursor, press SHIFT X.

Pressing D W deletes characters from the cursor to the end of the word on which it is located. Typing ***n***D W deletes *n* words (*n* is the number of words you wish to delete).

To delete a line of text, press D twice. To delete a specified number of lines, type ***n***D D (*n* is the number of lines you wish to delete). Pressing SHIFT D from any position in a line deletes all the line located to the right of the cursor.

Type D T*c* to delete all characters up to the character *c*, where *c* is any keyboard character. To delete a marked block of text, see the "Handling Blocks of Text" in this section.

Changing Text

Several change commands let you replace a specified amount of text with any amount of new text. When using these commands, press BREAK to end the change procedure.

To replace 1 character with another, position the cursor over the character to be changed and press R. Then type the replacement character. You can replace more than 1 character by typing SHIFT R together. Then type the replacement characters. To end the replace session, press BREAK.

Do not use the *replace* function to replace a block of text with a greater amount of text. If you continue replacing characters beyond the original line, the cursor moves to the next line and continues replacing characters. If you continue replacements beyond the end of a procedure. D.L. LOGO continues the replace into the next procedure, if any exists. Press BREAK to exit the replace mode.

Press C W to change the word on which the cursor is located. Typing *n*C W changes *n* words (*n* is the number of words to change).

To change an entire line, press C C. To change several lines, type *n*C C, (*n* is the number of lines to change). Pressing SHIFT C at any position in a line lets you change the remainder of that line.

You can also change characters up to a specified character. To do this, type C T*c* (*c* is the character on which the change ends). To change blocks of text, see the block commands later in this section.

A Powerful Yank

Yank is an editing feature that can save you a lot of time. This function lets you move copies of words or lines to a new position in your procedure or even to another procedure in the workspace.

To demonstrate the Yank function, type or load the SPIN procedure used in Chapter 3:

```
TO SPIN
     SETSPLIT 1
     PU FD 20 PD
     REPEAT 36 [REC]
END
```

Now position the cursor at the beginning of Line 3:

```
_PU FD 20 PD
```

and type:

```
[O] [→] SETPC 3 [ENTER]
[→] REPEAT 4 [
```

Pressing [O] opens a space to let you insert the 2 new lines. To exit the insert mode, press [BREAK].

Move the cursor to the last line of the procedure and position it over the E in End. Type:

```
[I] [→] [ENTER]
[BREAK]
```

Now move the cursor to the beginning of Line 4:

```
_ETPC 3
```

and press [Y] [W].

There is no change to the screen at this time. Move the cursor to the blank line you created and type:

SHIFT P

The word SETPC appears at the cursor position. Use SHIFT and → to move the cursor 1 space past the end of the line and type **I PC - 1]** BREAK. The line should look like this:

```
SETPC PC - 1]
```

The completed procedure should be the same as the example below:

```
TO SPIN
     SETSPLIT 1
     PU FD 20 PD
     SETPC 3
     REPEAT 4[
     REPEAT 36 [REC]
     SETPC PC - 1]
END
```

To see the results of your changes, use the REC procedure (as demonstrated in Chapter 3) in conjunction with the SPIN procedure.

The Big Move

As well as letting you move, insert, and delete words and lines, D.L. LOGO lets you manipulate blocks of program or procedure text.

First mark the block with the MB, ME, and MM combination of keys. MB marks the beginning of a block. ME marks the end of a block. MM moves the marked block. Try the procedure by moving the cursor to Line 2 in the SPIN procedure:

```
TO SPIN
SETSPLIT 1
_U FD 2Ø PD
```

Press [M] [B].

Move the cursor to the end of Line 5 of the same procedure:

```
TO SPIN
SETSPLIT 1
PU FD 2Ø PD
SETPC 3
REPEAT 4[
REPEAT 36 [REC]_
```

Press [M] [E].

Now type [SHIFT] **G** to move the cursor to the end of the program, then type:

[M] [M]

The marked text is moved to the new location on the screen, beginning at the cursor position. You can move or duplicate blocks of procedures anywhere in the work space. Experiment with these functions, you can use them to extract particular routines from 1 procedure for use in another or to break a large procedure into smaller, easier-to-handle, procedures.

Because the block move was only for demonstration purposes, type [U] to *undo* the block move.

Section 2
The Search

D.L. LOGO's search and replace features aid you in quickly finding a particular occurrence of a name, list or character. You can replace any characters with other characters.

To see how it works, type some procedure that uses several FD commands. Then, from any position in the workspace, type:

```
/FD [ENTER]
```

The cursor jumps to the first occurrence of the primitive FD. To find the next occurrence, press:

```
N
```

and the cursor jumps to the next FD occurrence. You can use [N] to repeat any search command.

Making the Change

Replace is accomplished in conjunction with the Search function. To replace the first FD primitive with the BK primitive, type:

```
/FD/BK [ENTER]
```

To cause a replacement of the next occurrence of FD, press [N]. To replace all FD primitives with BK, type:

```
/FD/BK/G [ENTER]
```

When you type /G at the end of a search and replace command, D.L. LOGO replaces all occurrences of the indicated characters with the replacement text.

. . . . About Special Editing Features

Many special editing functions do not save time or effort on short procedures. Often these features are not useful unless you are writing larger and more complex procedures and programs.

. . . . About Changes

When making global changes to a large program or procedure, make a note of any values you plan to change or that might be changed inadvertently. Global changes often affect more than you expect.

If your procedure contained occurrences of the number 40, you can replace them with the number 20 by typing:

```
/40/20/G ENTER
```

Only The Beginning

This section describes only some of D.L. LOGO's advanced editing features. The following chart describes all editing operations.

Section 3
Edit Commands Reference

Editor Cursor Control

Keys to press	Result
↓ or J	moves the cursor down 1 line.
↑ or K	moves the cursor up 1 line.
→ or L	moves the cursor right 1 character.
← or H	moves the cursor left 1 character.
SHIFT ↑	moves the cursor back 1 page.
SHIFT ↓	moves the cursor forward 1 page.
SHIFT → or SHIFT $	moves the cursor to the end of a line.
SHIFT ← or SHIFT 0	moves the cursor to the beginning of a line.
W	moves the cursor forward 1 word.
SHIFT W	moves the cursor to the first character after the next space.
B	moves the cursor to the beginning of the previous word.
SHIFT B	moves the cursor to the previous word surrounded by spaces.
E	moves the cursor to the end of the current or next word.

. . . . *About Editing Files*

Although this chapter deals with editing LOGO procedures, D.L. LOGO's comprehensive editor can also be used to edit data files. If you create a data management program for LOGO, make use of this facility rather than writing an editing procedure.

. . . . About Surrounding Spaces

Some of the editing features described in this section act only on words surrounded by spaces. *This distinction is made because some of the words you type in a procedure may not have spaces around them, such as:*

PRINT"HELLO

In this case, functions that search for a word surrounded by spaces, will not find the word HELLO.

Editor Cursor Control

Keys to press	Result
SHIFT E	moves the cursor to the end of the current word or the next word surrounded by spaces.
G	moves the cursor to the beginning of the workspace.
SHIFT G	moves the cursor to the end of the workspace.

Insert Commands (insert commands are terminated by BREAK)

Keys to press	Result
I	inserts at the cursor position.
SHIFT I	inserts at the beginning of the current line.
A	inserts after the current cursor position.
SHIFT A	inserts at the end of the current line.
O	inserts after the current line.
SHIFT O	inserts before the current line.

Delete Commands

Keys to press	Result
X	deletes character under cursor.
n X	deletes the next *n* characters.

Delete Commands

Keys to press	Result
SHIFT X	deletes the character before the cursor.
D W	deletes the current word.
n D W	deletes the next *n* words.
D SHIFT W	deletes the next block of text surrounded by spaces.
n D SHIFT W	deletes the next block of text containing *n* words surrounded by spaces.
D D	deletes the current line.
n D D	deletes the next *n* lines.
SHIFT D	deletes the remainder of the current line.
D T *x*	deletes to the first occurrence of character *x*.
D M	deletes a marked area (see Special Commands for information on marking areas of text).

Change Commands

Keys to press	Result
R	replaces the character currently under the cursor.
SHIFT R	replaces characters until you press BREAK.
C W	replaces the current word.
n C W	replaces the next *n* words.
C SHIFT W	replaces the next word surrounded by spaces.
n C SHIFT W	replaces the next *n* words that are surrounded by spaces.
C C	changes the current line.
n C C	changes the next *n* lines.
SHIFT C	changes the remainder of the current line.
C T *x*	changes the following text to the next occurrence of character *x*.
C M	changes the marked area (see Special Commands for information on marked areas).

Yank Commands

Keys to press	Result
Y W	yanks the current word.
n Y W	yanks the next *n* words.
Y SHIFT W	yanks the next word that is surrounded by spaces.
n Y SHIFT W	yanks *n* words surrounded by spaces.
Y Y	yanks the current line.
n Y Y	yanks the next *n* lines.
SHIFT Y	yanks the remainder of the line.
Y T *x*	yanks to the next occurrence of the character *x*.

Search and Replace

Keys to press	Result
/xxxx	searches for the next occurrence of characters *xxxx*.
/xxxx/yyyy	searches for the next occurrence of *xxxx* and replaces the characters with *yyyy*.
/xxxx/yyyy G	replaces all occurrences of *xxxx* with characters *yyyy*.
N	repeats last search/change.

Special Commands

Keys to press	Result
U	undoes the last change.
N	repeats the last change.
M B	marks the beginning of a block of text.
M E	marks the end of a block of text.
M M	moves the marked block of text.
M SHIFT P	copies the marked block of text.
SHIFT P	inserts a copy of the last deleted or yanked block of text immediately after the current cursor position.
P	inserts a copy of the last deleted or yanked block of text immediately *at* the current cursor position.
%	positions the cursor on the parenthesis matching the current parenthesis.
CTRL 0	inverts character case.
SHIFT J	joins 2 lines.
?	shows text buffer information.

APPENDIX A

Glossary of Terms

access	To get to information stored in your computer or other device.
append	To add a file from diskette to the existing contents of D.L. LOGO's workspace.
argument	Data you supply with a primitive to direct the operation of that primitive.
background color	The overall graphics screen color, set by the BACKGROUND primitive.
backup	A copy of a program, a file, or all the information on a diskette.
boot	To start a disk operating system. You *boot* OS-9 before loading D.L. LOGO.
bottom-end loop control	The control of a loop process by a command following the loop.

bug	A program error that causes a procedure to stop or act differently than the programmer intends.
byte	The basic unit of information for a computer. For example, a byte can contain the information to produce a single character on a video screen or a printer.
Central Processing Unit	The circuits in a computer that control the execution of instructions.
character	A symbol that can be produced on a video display screen or printer.
code	Special characters to which D.L. LOGO can respond. For instance, you must convert musical scores into code for D.L. LOGO. D.L. LOGO then converts the code into musical tones.
command	A sequence of characters (primitives and arguments) that tells LOGO to take a particular action. A command can be typed at the keyboard or can be part of a procedure.
condition test	A programming process that determines whether a condition is true or false. The result of the test determines the next program operation.

CPU	The common abbreviation for Central Processing Unit.
current command	The command presently being executed.
current directory	The diskette area where you are currently operating. SAVE or LOAD primitives automatically select this area unless you specify a different directory.
cursor	A character that indicates the place that the next typed character will appear. In the immediate mode, the cursor is a blue rectangle. In the graphics text mode, the cursor is a single underline character.
data	Items of information (programs, procedures, or text) that a computer can generate or process.
debug	To find and eliminate program errors.
default	A command or operation value that is automatically established by a program but can be changed.
direct access	The ability to read from, or write to, an element in a file without reading any preceding elements.
directory	A portion of a diskette set aside for the storage of files. Each directory has a name that you use to access the directory.

disk drive	The mechanical and electronic assembly that handles the storage and retrieval of information to and from a diskette.
diskette	The flexible platter, made of a plastic-like material, on which your computer magnetically stores data.
display	The video screen (TV or monitor).
dots	The colon symbol (:) preceding a variable name.
edit	The process of modifying text. You edit when you add, change, or delete portions of a procedure.
editor	A program that lets you use specific commands to modify text.
element	A unit within a LOGO word, list, or sentence.
erase	To delete a block of text, a procedure, or a program from D.L. LOGO's workspace or a file from a diskette.
error code	The numeric value given a particular D.L. LOGO error and error message.
error message	Text that appears on the screen and indicates that something is wrong with the operation of a procedure, your computer, or a peripheral.

exit	To leave a procedure or program. You can exit to a higher or lower program, or to the immediate mode from a procedure or program.
file pointer	An index that contains the positions of data within a file.
file	A block of information your computer uses for a particular function. A file can contain an operating system (such as OS-9), a language (such as D.L. LOGO), or text.
filename	A name that identifies a block of data saved to a diskette as a file.
foreground color	The color used by D.L. LOGO to create graphics.
format	To organize a diskette into tracks and sectors for the storage of files. Also, the visual arrangement of a procedure in the workspace or of text on the screen.
global	An all-inclusive operation. For instance, a *global search* examines all text in the workspace.
graphics mode	The portion of D.L. LOGO that displays the results of graphics commands.
graphics	Lines, dots, and shapes produced on the screen by D.L. LOGO's primitives.

hardcopy	A printer copy of computer data (text or graphics).
home	The exact center of the graphics screen.
immediate mode	An operation in D.L. LOGO that lets you type commands or arithmetic functions for immediate execution.
input	The flow of data from a device (such as a disk drive or printer) to your computer.
interface	1- or 2-way communication between 2 devices, such as a computer and a disk drive.
keyboard buffer	A portion of your computer's memory where the values of the keys you press are stored. These values are used to produce characters on the screen, or control D.L. LOGO functions, or both.
language	A set of instructions (commands) that your computer interprets to perform tasks. BASIC and D.L. LOGO are 2 languages you can use to control your computer.
layout	The organization of a display screen. The arrangement of graphics, text, or both.
link	To cause the execution of a procedure from another procedure.

list	A type of variable that consists of 1 or more words, 1 or more lists, or of a combination of words and lists.
load	To transfer a file from diskette into your computer's workspace.
loop	One or more procedure statements that are successively repeated.
member	A unit within a LOGO word, list, or sentence.
memory	The portion of your computer system that stores data or values.
menu	A screen display that gives you a list of options.
mixed number	A numeric value with a decimal fractional part.
mode	A particular function of D.L. LOGO.
nest	To place 1 or more loops (repetitive processes) within another loop or loops.
operating system	A set of associated programs that directs your computer's operation.
output	The flow of data from your computer to another device, such as a disk drive or a printer.

parameter	Data that you supply with a primitive to direct the operation of that primitive.
pen color	The color currently used to create graphics.
precision	The accuracy of a calculation. Precision is defined by the maximum number of digits calculated after a decimal point.
primitive	A built-in command that causes LOGO to perform a particular action.
procedure list	An action or actions (enclosed in square brackets) that LOGO executes when a defined condition is met.
procedure name	The label you give a procedure when it is created.
program	A set of instructions that tells a computer how to perform a task.
prompt	A screen display that requires some action, such as pressing a key.
recursion	A function that causes a procedure to repeat its execution.
remote control	To control a device with your computer.
replace	To replace specified text with other text using D.L. LOGO's editor.

restore	(1) The process of returning a file from diskette to D.L. LOGO's workspace. (2) The process of reversing an editing command.
reverse characters	Lowercase characters (displayed as green characters surrounded by a black background).
ROM	The common abbreviation for Read Only Memory. Integrated circuit chips containing data that your computer can read, but cannot change.
save	To transfer data from your computer to diskette.
screen boundary	The visible limits of your TV or monitor screen.
screen dump	A program that uses a printer to reproduce graphic images on paper.
screen page	Text that fills the display screen from top to bottom (16 lines).
search	To look through D.L. LOGO's workspace for occurrences of specified text using D.L. LOGO's editor.
sector	One of several units of storage within a diskette track.
sentence	A LOGO list with the first level of brackets removed.

SHELL	The OS-9 program that reads and interprets commands.
single command mode	An operation in D.L. LOGO that lets you type commands or arithmetic functions for immediate execution. Also referred to as immediate mode.
split value	The number of lines set at the bottom of the graphics screen for text displays.
step	One unit of movement on the graphic screen.
string	A series of characters.
syntax	The form of a command, including the order of the arguments and the way the command must be stated.
toggle	To switch between 2 conditions, such as on and off.
top-end loop control	The control of a loop process by a command statement preceding the loop.
track	A physical unit of storage on a diskette.
turtle	The graphics character that moves around the screen. The turtle can be *hidden* (invisible) or *shown* (visible), and can create graphics as it moves (in a pendown state) or move without creating graphics (in a penup state).

variable	A portion of your computer's memory that you establish to store data. The contents of a variable can be changed.
voice	Any of the melodic parts of a music composition.
word	A type of variable that consists of 1 or more characters.
workspace	An area in your computer's memory that D.L. LOGO uses to store procedures and programs.
wraparound	The action D.L. LOGO takes when the line you are typing reaches the edge of the screen, the text automatically continues one line down, at the left of the screen.

APPENDIX B

Sample D.L. LOGO Programs

The programs in this appendix provide additional examples of D.L. LOGO at work. Some are useful in maintaining a D.L. LOGO library, some have education or business applications, and some are only for your enjoyment.

To add any of these programs to your LOGO library, enter the edit mode and type the listing as shown. When finished, enter the immediate mode and save the program by typing **SAVE** ***"programname*** ENTER. The word *programname* is used to represent the actual name of the program you are saving. For instance, after typing the following CAT program, save it by typing **SAVE "CAT** ENTER.

When any of these programs are in the D.L. LOGO workspace, (typed or loaded from diskette), execute them by typing ***programname*** ENTER. Again, *programname* represents the actual name of the program you wish to execute.

Information on how to type programs, enter and exit the edit and immediate modes, and save and load programs is contained in the first 4 chapters of this manual.

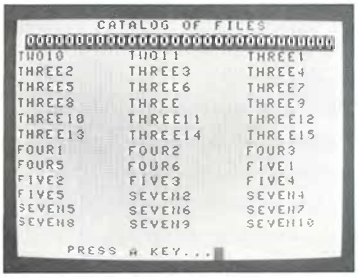
CAT
A Disk Catalog Program

```
TO CAT
     MAKE "BLANK LIST " " " " " " " " "
      "
     MAKE "CT 0
     MAKE "CX 0
     CLEARTEXT
     MAKE "TITLE SE [\ \ \ \ \ \ \
      CATALOG OF FILES] CHAR 13 [
      ooooooooooooooooooooooooooooooo]
     PRINT :TITLE
     SETCURSOR 3 0
     PRINT1 [\ OUTPUT TO PRINTER? Y/
      N...]
     MAKE "PRINT RC
     SETCURSOR LINE +1 0
     PRINT1 [\ DRIVE NUMBER?...]
     MAKE "DRIVE RC
```

. . . . About CAT

The CAT program provides an easy-to read listing of diskette directories. When you start the program, the prompt, OUTPUT TO PRINTER? Y/N..., appears. Press Y if you wish a hardcopy (a paper copy from your printer) or N if you wish the diskette contents displayed only to the screen.

The prompt, DRIVE NUMBER?..., then appears. Put the diskette you wish to examine in one of your disk drives and press the number of that drive. For example, if your diskette is in D1 (Drive 1), press 1*. CAT reads all the files from the current directory and displays them in 3 columns on the screen. If you have selected printer output, the files are also listed in 3 columns on your printer. When the screen is full, the display stops and waits for you to press a key. Press* SPACEBAR *to continue the listing. Use the CHD command before using CAT to examine other directories.*



```
           CATALOG OF FILES
00000000000000000000000000000000000000
TWO10          TWO11          THREE1
THREE2         THREE3         THREE4
THREE5         THREE6         THREE7
THREE8         THREE          THREE9
THREE10        THREE11        THREE12
THREE13        THREE14        THREE15
FOUR1          FOUR2          FOUR3
FOUR5          FOUR6          FIVE1
FIVE2          FIVE3          FIVE4
FIVE5          SEVEN2         SEVEN4
SEVEN5         SEVEN6         SEVEN7
SEVEN8         SEVEN9         SEVEN10

       PRESS A KEY...
```

```
        IF :PRINT="Y [COPYON] ELSE [
         COPYOFF]
        MAKE "C CATALOG WORD "\/D :DRIVE
        MAKE "END COUNT :C
        CLEARTEXT
        PRINT1 :TITLE
        PRINT
        DO [NEXT]
        WHILE :CT<:END
        COPYOFF
        PRINT
        PRINT1 [PRESS A KEY...]
        MAKE "NUM RC
        CAT
END

TO NEXT
        MAKE "CT :CT+1
        MAKE "CX :CX+1
        SELECT [
                :CX<3 [
                        MAKE "LENGTH 12-COUNT
                         ITEM :CT :C
                        PRINT1 ITEM :CT :C
        PRINT1 PIECE 1 :LENGTH :BLANK]
                :CX=3 [
                        PRINT1 ITEM :CT :C
                        MAKE "CX 0 PRINT]]
        IF LINE=14 [
                COPYOFF
                SETCURSOR 15 5
```

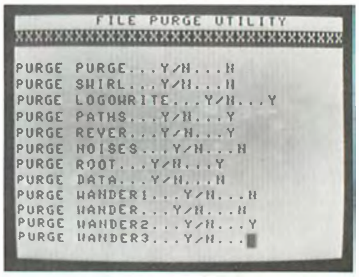
```
          PRINT1 [PRESS A KEY...]
          MAKE "NUL RC
          CLEARTEXT
          PRINT1 :TITLE
          PRINT]
     IF :PRINT="Y [COPYON]
END
```

PURGE To Delete Files

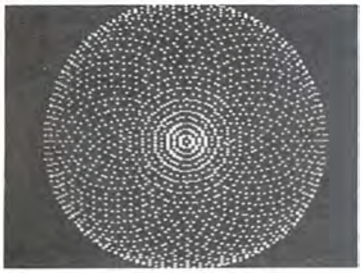
```
TO PURGE
     CLEARTEXT
     MAKE "PURGELIST []
     SETCURSOR 0 5
     PRINT [FILE PURGE UTILITY]
     PRINT "xxxxxxxxxxxxxxxxxxxx
      xxxxxxxxxxx
     SETCURSOR 3 0
     MAKE "FILES CATALOG
     MAKE "END COUNT :FILES
     FOR "I 1 :END 1 [
          PRINT 1 [PURGE\ ] ITEM :I
           :FILES [...Y\/N...]
          MAKE "CHECK RC
          IF :CHECK="Y [MAKE "PURGELIST
           (LPUT ITEM :I :FILES
           :PURGELIST)]
          PRINT]
     MAKE "END COUNT :PURGELIST
```

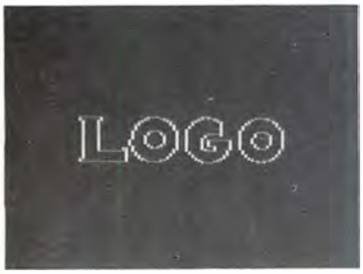
*.... **About the Purge Program***

This program reads all the filenames from the current directory, displays each filename, and asks you if you wish to delete it. To cause a file to be deleted, press Y *at the prompt. If you do not wish a file to be deleted, press* N*. After you answer Y or N for each file, PURGE displays the names of the files it erases.*



```
          FILE PURGE UTILITY
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

PURGE PURGE...Y/N...N
PURGE SWIRL...Y/N...N
PURGE LOGOWRITE...Y/N...Y
PURGE PATHS...Y/N...Y
PURGE REVER...Y/N...Y
PURGE NOISES...Y/N...N
PURGE ROOT...Y/N...Y
PURGE DATA...Y/N...N
PURGE WANDER1...Y/N...N
PURGE WANDER...Y/N...N
PURGE WANDER2...Y/N...Y
PURGE WANDER3...Y/N...█
```





```
        FOR "I 1 :END 1 [
                PRINT [ERASING] ITEM :I
                 :PURGELIST
                ERASEFILE ITEM :I :PURGELIST]
END
```

GRAPHICS

```
TO SWIRL
        FULLSCREEN
        CS HT PENUP
        REPEAT 36 [RT 10
                REPEAT 72 [FD 4 RT 5 DOT XCOR
                 YCOR]]
        SPLITSCREEN
END
```

LOGO LETTERS

```
TO LOGOWRITE
        CS HT
        SETXY XCOR-65 YCOR-10
        L
        SETX XCOR+45
        O
        SETXY XCOR+19 YCOR+5
        SETH 0
        G
        SETXY XCOR+11 YCOR-11
        O
END
```

```
TO L
        SERIF
        FD 25
        SERIF
        RT 90 FD 10 RT 90
        SERIF
        FD 20 LT 90 FD 15
        SERIF
        RT 90 FD 7 RT 90 FD 27
END

TO O
        REPEAT 91 [FD 1 RT 3.956]
        PU
        RT 90 FD 9 LT 90
        PD
        REPEAT 36 [FD 1 RT 10]
END

TO G
        REPEAT 31 [FD 1 RT 3.956]
        PU
        REPEAT 15 [FD 1 RT 3.956]
        PD
        REPEAT 45 [FD 1 RT 3.956]
        PU
        REPEAT 45 [FD 1 RT 3.956]
        PD
        RT 90 FD 15
        LT 90 FD 3 LT 90 FD 4
```

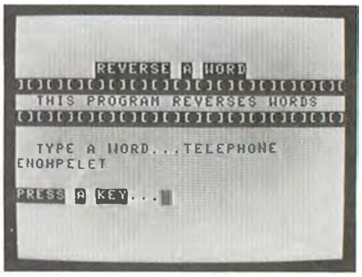
```
     RT 90 FD 3 RT 90 FD 3
     REPEAT 10 [FD 1 RT 12]
     REPEAT 15 [FD 1 RT 7]
     LT 130 FD 8
END

TO SERIF
     LT 90 FD 2 RT 90 FD 2 RT 90 FD 2 LT
      90
END
```

TWISTING TRIANGLE

```
TO PATHS
     FULLSCREEN
     CS MAKE "X 0
     REPEAT 53 [SETPC 1
          REPEAT 3 [FD :X RT 121
               IF PC<3 [SETPC PC+1]
               MAKE "X :X+3]]
     SPLITSCREEN
END
```

REVERSE A WORD

```
TO REVER
     CLEARTEXT
     SETCURSOR 3 8
     PRINT [reverse a word]
     PRINT "XXXXXXXXXXXXXXXXX
      XXXXXXXXXXXXXXX
     SETCURSOR 5 2
     PRINT [THIS PROGRAM REVERSES WORDS]
     PRINT "XXXXXXXXXXXXXXXXX
      XXXXXXXXXXXXXXX
     SETCURSOR 8 2
     PRINT1 [TYPE A WORD...]
     MAKE "W FIRST RQ
     MAKE "C COUNT :W
     FOR "X :C 1 -1[
     PRINT1 (ITEM :X :W)]
     PRINT
     PRINT
     PRINT1 [press a key...]
     MAKE "NULL RC
     REVER
END
```



```
        REVERSE A WORD
][][][][][][][][][][][][][][][][][][
  THIS PROGRAM REVERSES WORDS
[][][][][][][][][][][][][][][][][][]

  TYPE A WORD...TELEPHONE
ENOHPELET

PRESS A KEY...
```

NOISES

```
TO FALL
     FOR "I 4000 100 -50
     [SOUND :I 40]
     SOUND 400 100
     FOR "I 800 400 -100 [SOUND :I 10]
     SOUND 200 200
     MAKE "T 400
     REPEAT 10 [SOUND :T 1
          MAKE "T :T-40]
END

TO ZAP :L
     FOR "I 1000 6000 100
          [SOUND :I :L]
     FOR :I 5000 1000 -100
          [SOUND :I :L/10]
END

TO BOMBER
     WHILE "TRUE
          [SOUND 700 2]
END

TO THUMP
     MAKE "T 200
     REPEAT 50 [
          SOUND :T 100
          MAKE "T :T-.1]
END
```

ROOTS

```
TO ROOT :N :HI :LO
      MAKE "T (:HI+:LO)/2
      IF :T↑3>:N+.01 [OUTPUT ROOT :N :T
       :LO]
      IF :T↑3<:N-.01 [OUTPUT ROOT :N :HI
       :T]
      OUTPUT :T
END
```

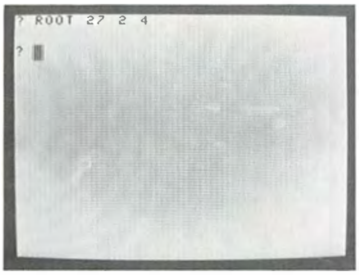
WANDERING TURTLE

```
TO PATH
      CS FULLSCREEN
      SETXY 48 -35
      REPEAT 4 [DOODLE 140 LT 90]
END

TO DOODLE :N
      IF :N<4 [STOP]
      DOODLE :N/4
      RT 90 FD 4
      DOODLE :N/4
      LT 90 FD 4
      DOODLE :N/3
      LT 90 FD 4
      DOODLE :N/3
      RT 90 FD 4
      DOODLE :N/4
END
```

. . . . *About Root*

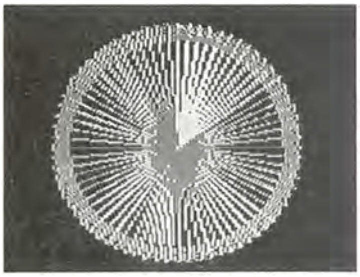
ROOT determines the cube root of a number to an accuracy of plus or minus 0.01. You can set a higher or lower accuracy by using different values in place of .01 in lines 3 and 4. A lower value (for instance .001) gives higher accuracy but takes longer to calculate. A higher value (for instance .1) gives less accuracy, but the calculation is quicker. You can also change the program to calculate other root values by changing the 3 in Lines 3 and 4 to the root you wish to calculate.

To use the program, type the procedure name, the number for which you wish to find the cube root, a high guess, and a low guess. The closer your high and low guesses come to the actual cube root, the faster the calculation is performed. For instance, to calculate the cube root of 100 you might determine that 5^3 is 125 and 4^3 is 64. The cubed root of 100 must be between 4 and 5. Thus, to perform the calculation, type ***ROOT 5 4*** $\boxed{\text{ENTER}}$.


```
? ROOT 27 2 4
?
```

TRIANGULAR RING

```
TO TRIRING
     SETBG 12 SETPC 1 CS
     REPEAT 72 [SETPC PC +1
     REPEAT 3 [FD 80 RT 120 SETPC PC+1]
     RT 5 SETPC 0]
END
```

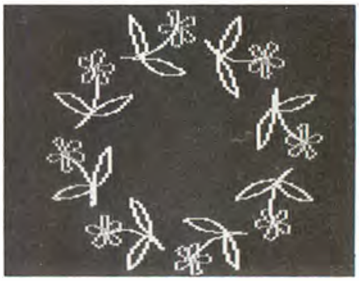


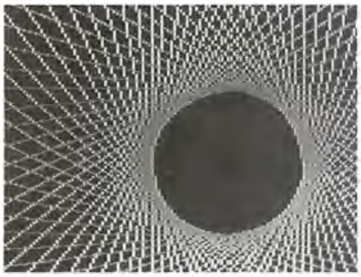
OCTAGONS

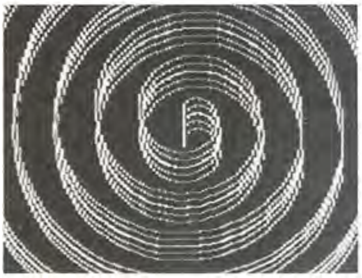
```
TO OCT
     CS HT
     FULLSCREEN
     SETPC 1
     SETXY -149 90
     REPEAT 14 [ROW
          SETXY -146 YCOR-15
          SETPC PC+1
          IF PC=0 [SETPC 1]]
END

TO ROW
     REPEAT 20 [SETX XCOR+13 DRAW]
END

TO DRAW
     REPEAT 6 [FD 15 RT 60]
END
```

RING OF FLOWERS

```
TO WREATH
     CS
     FULLSCREEN
     SETXY -60 55
     REPEAT 8 [REP PU RT 249 FD 80 PD ]
END

TO REP
     HT
     SETPC 1
     REPEAT 5 [PETAL]
     SETPC 3
     REPEAT 11 [BK 4 RT 7]
     REPEAT 3 [FD 4 LT 7]
     RT 10
     LEAF LEAF
END

TO LEAF
     REPEAT 10 [FD 3 RT 4]
     RT 125
     REPEAT 10 [FD 3 RT 7]
END

TO PETAL
     RT 84 FD 10
     REPEAT 10 [RT 20 FD 1]
     FD 10
END
```







SPIDER'S WEB

```
TO WEB
     CS REPEAT 100 [ZIG]
END

TO ZIG
     RT 25 FD 250 RT 150 FD 250
END
```

COLORED RIBBONS

```
TO RIBBON
     CS MAKE "F 0
     REPEAT 3 [SETPC 0
          REPEAT 3 [ SETPC PC+1
          HOME
          MAKE "F :F+5 FD :F SPIR]]
END

TO SPIR
     FULLSCREEN
     MAKE "R 5
     REPEAT 400 [ FD :R*.07 RT 5
          MAKE "R :R+.48]
END
```

SPIRAL

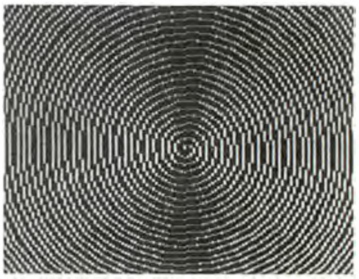
```
TO SPIR
     CS SETBG 12 CS
     FULLSCREEN
     MAKE "R 5
     REPEAT 1000 [FD :R*.07 RT 12 MAKE
      "R :R+.48]
     REPEAT 10 [REPEAT 16 [SETBG BG+1
      WAIT 3]
     SETBG 0]
     SETBG 12
END
```

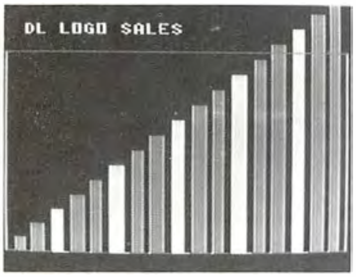
GRAPH

```
TO GRAPH
     SETCURSOR 3 1
     MAKE "VALUES []
     MAKE "CT 0
     MAKE "FACTOR 1
     INPUT
     BAR
     BOX
END

TO INPUT
     CLEARTEXT
     LABEL "START
     PRINT1 [NUMBER OF ELEMENTS...]
     MAKE "ELEM FIRST RQ
     IF :ELEM>17 [
```

. . . . About GRAPH

This program creates a bar graph from the data you supply. When you execute the program, it prompts you for the number of elements you wish to place in the graph (the number of bars that are produced). Type any number in the range of 1-17. You are asked for a title and then asked for a value (percentage) to assign each element of the graph. After you answer all the prompts, the program draws the graph, according to your specifications. If any of the values you give the elements are too large to fit the screen, the graph is automatically scaled down.



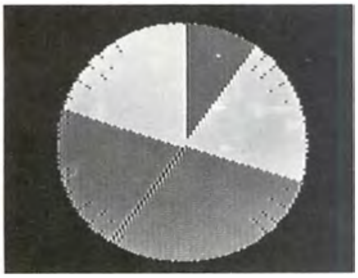


```
          PRINT [SORRY, TOO MANY
           ELEMENTS]
          GO "START]
     PRINT1 [TITLE OF GRAPH...]
     MAKE "TITLE RQ
     REPEAT :ELEM [
          MAKE "CT :CT+1
          PRINT1 [VALUE OF ELEMENT NO.]
           :CT [...]
          MAKE "VAL FIRST RQ
          IF :FACTOR<INT(:VAL/60) [MAKE
           "FACTOR INT :VAL/60]
          MAKE "VALUES LPUT :VAL
           :VALUES]
END
```

```
TO BAR
     HT
     CS
     MAKE "COLOR 0
     SETXY -110 -70
     MAKE "TEMP :VALUES
     REPEAT :ELEM [
          MAKE "COLOR :COLOR+1
          IF :COLOR>3 [MAKE "COLOR 1]
     SETPC :COLOR
     MAKE "HEIGHT FIRST :TEMP
     REPEAT 4 [FD :HEIGHT RT 90 FD 1 RT
      90 FD :HEIGHT LT 90 FD 1 LT 90]
     RT 90 FD 5 LT 90
     MAKE "TEMP BUTFIRST :TEMP]
END

TO BOX
     FD 135 LT 90
     FD (:ELEM*5)+(:ELEM*8)+5
     LT 90 FD 135 LT 90 FD 5
     SETXY -100 83
     TURTLETEXT :TITLE
END
```

*. . . . **About PIE***

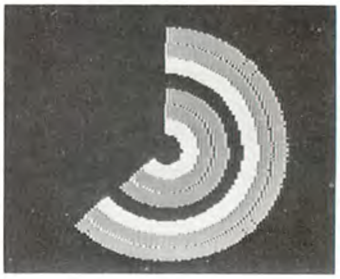
This program draws a pie graph with as many elements as you wish. It asks you how many elements (percentages) you wish to enter and the value of each element. After you give the required values, a pie chart is drawn in alternating red, white, and blue colors.



PIE

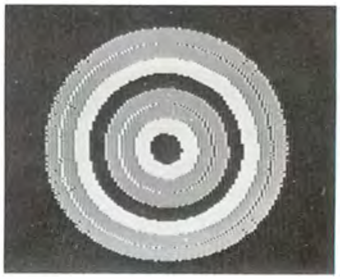
```
TO PIE
      CLEARTEXT
      MAKE "ITEM []
      MAKE "CT 1
      PRINT1 [NUMBER OF ENTRIES...]
      MAKE "NUMBER FIRST RQ
      INPUT
      DRAW
END

TO INPUT
      REPEAT :NUMBER [
            PRINT1 [NO..] :CT [\ PERCENT...]
            MAKE "QUANTITY FIRST RQ
            MAKE "ITEM SE LPUT :QUANTITY
             :ITEM
            MAKE "CT :CT+1]
END
```

```
TO DRAW
     FULLSCREEN
     CS
     SETPC 1
     REPEAT :NUMBER [
          MAKE "AMOUNT FIRST :ITEM
          SETPC PC+1
          REPEAT :AMOUNT*3.6 [FD 80 BK
           80 RT 1]
          IF PC=3 [SETPC 0]
          MAKE "ITEM BUTFIRST :ITEM]
END
```



BULL'S EYE

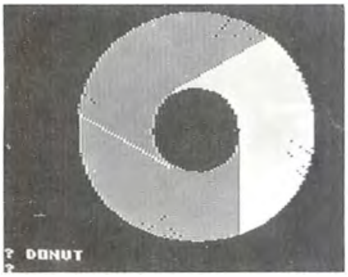
```
TO BULL
     CS
     FULLSCREEN
     REPEAT 360 [SETPC 4
          REPEAT 8 [FD 10 SETPC PC-1
          IF PC<0 [SETPC 3]]
     REPEAT 8 [SETPC PC+1 PU BK 10 PD
          IF PC<3 [SETPC 0]]
     RT 1]
END
```



. . . . About GUESS

GUESS is a version of a popular memory game. The program draws 12 boxes on the screen and each has a randomly selected name that matches 1 other box.

To view the name of a box you type its number. Then, if you can type the number of the corresponding box, a match is recorded. If you guess wrong, the names of the 2 boxes are erased. When all 6 matches are successfully made, the game ends, and your number of tries are displayed.



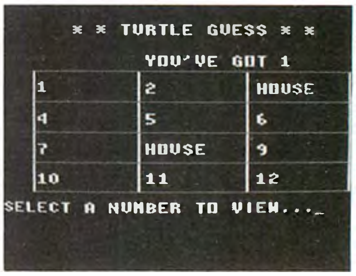
DONUT

```
TO DONUT
    CS SETPC 1
    SETXY 30 -60
    REPEAT 3 [REPEAT 120 [
        FD 70 RT 1 BK 70.5]
        SETPC PC+1]
END
```

A GUESS

```
TO GUESS
    SETBG 14
    CATCH 6 [ERR] CATCH 5 [ERR] CATCH 2
     [ERR]
    HT SETSPLIT 5 CLEARTEXT
    MAKE "GOT 0
    MAKE "CHOICES []
    MAKE "TRYS 0
    CS GRID SETUP NUMBER PICK
END
```

```
TO GRID
     SETPC 1
     SETXY -105 -30
     REPEAT 4 [RT 90 FD 210
          LT 90 FD 20 LT 90
          REPEAT 3 [FD 70 RT 90 BK 20 FD
           20 LT 90]
          RT 90]
     SETPC 3
END

TO SETUP
     MAKE "POS [[-102 43][-32 42][40 42]
      [-102 22][-32 22][40 22][-102 02]
      [-32 02][40 02][-102-18][-32-18]
      [40 -18]]
     MAKE "CHOICES SHUFFLE [$100 $100
      CAR CAR BOAT BOAT $5 $5 PEN PEN
      HOUSE HOUSE]
     FOR "I 1 12 1 [MAKE ITEM :I
      :CHOICES "BLANK]
END
```



```
        * * TURTLE GUESS * *
            YOU'VE GOT 1
 1         2         HOUSE
 4         5         6
 7         HOUSE     9
 10        11        12
SELECT A NUMBER TO VIEW..._
```

```
TO NUMBER
     FOR "I 1 12 1 [MAKE "X FIRST ITEM
      :I :POS
          MAKE "Y LAST ITEM :I :POS
          SETXY :X :Y
          IF (THING ITEM :I
           :CHOICES)<>"GOTIT [TURTLETEXT
           :I "\ \ \ ]]
END
```

```
TO PICK
     MAKE "TRYS :TRYS+1
     CLEARTEXT
     CLEAR
     GETONE
     IF (THING ITEM :NUM1
      :CHOICES)="GOTIT [ERR]
     CHECK
     TURTLETEXT :VIEW
     MAKE "ONE :VIEW
     CLEARTEXT
     GETTWO
     IF (THING ITEM :NUM2
      :CHOICES)="GOTIT [ERR]
     CHECK
     TURTLETEXT :VIEW
     MAKE "TWO :VIEW
     IF :ONE=:TWO [DELETE]
     WAIT 100
     NUMBER
     PICK
END
```

```
TO DELETE
     SETXY -30 60
     MAKE ITEM :NUM1 :CHOICES "GOTIT
     MAKE ITEM :NUM2 :CHOICES "GOTIT
     MAKE "GOT :GOT+1
     SAY [YOU'VE GAUHT] :GOT
     TURTLETEXT [YOU'VE GAUHT] :GOT
     IF :GOT=6 [FINISH]
     PICK
END

TO CHECK
     IF (ANYOF :NUM1>12 :NUM1<1 :NUM2>12
      :NUM2<1) [ERROR 6]
     SETXY (FIRST :PLACE)+3 LAST :PLACE
END

TO ERR
     SOUND 300 200
     SETXY -75 80
     TURTLETEXT [ENTRY NOT RECOGNIZED]
     WAIT 50
     NUMBER PICK
END
```

```
TO GETONE
     SOUND 700 200
     PRINT1 [SELECT A NUMBER TO VIEW...]
     MAKE "NUM1 FIRST RQ
     SAY :NUM1
     MAKE "PLACE ITEM :NUM1 :POS
     MAKE "VIEW ITEM :NUM1 :CHOICES
END
TO GETTWO
     PRINT1 [SELECT ANOTHER NUMBER...]
     MAKE "NUM2 FIRST RQ
     SAY :NUM2
     MAKE "VIEW ITEM :NUM2 :CHOICES
     MAKE "PLACE ITEM :NUM2 :POS
END

TO CLEAR
     SETXY -75 80
     TURTLETEXT [** TURTLE GUESS **]
     MAKE "NUM1 1
     MAKE "NUM 2 1
END
```

```
TO FINISH
     CLEARTEXT
     PRINT [YOU GOT THEM ALL!]
     PRINT [NUMBER OF TRIES=] :TRYS
     SAY [YOU'VE GAUHT THEM ALL, THAT
      WAS FUN]
     FOR "I 1 12 1 [MAKE THING ITEM :I
      :CHOICES "BLANK]
     PRINT1 [PLAY AGAIN? Y\/N...]
     MAKE "GAME RC
     IF :GAME="Y [GUESS] ELSE [CLEARTEXT
      TEXTSCREEN PRINT [THANKS]
      TOPLEVEL]
END
```

APPENDIX C
Primitive Reference

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
ABS Chap. 9	—	Returns the absolute value of a given number.
ALLOF Chap. 9	—	Performs a logical AND operation on a series of true/false operations or statements.
ANYOF Chap. 9	—	Performs a logical OR operation on a series of true/false operations or words.
APPEND Chap. 4	—	Loads the specified diskette file below the procedure or procedures currently in the workspace.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
ASCII Chap. 9	—	Returns the ASCII code for a specified character.
ATAN Chap. 9	—	Calculates the arctangent of a specified ratio.
BACK Chap. 1	BK	Moves Turtle backward a specified number of steps.
BACKGROUND Chap. 2	BG	Displays the current background color.
BUTFIRST Chap. 8	—	Extracts all but the first element of an object.
BUTLAST Chap. 8	—	Extracts all but the last element of an object.
BUTTON? Chap. 12	—	Determines if you press the specified joystick button.
BYE Chap. 1	—	Causes D.L. LOGO to exit to OS-9.
CATALOG Chap. 4	—	Displays all diskette files.
CATCH Chap. 13	—	Redirects LOGO's normal error function.
CHAR Chap. 9	—	Returns the character whose ASCII value equals the specified number.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
CHD– Chap. 4	—	Changes the current LOGO directory to the specified directory.
CLEAN Chap. 2	—	Clears the graphics screen without moving current Turtle coordinates.
CLEARINPUT Chap. 3	—	Clears the keyboard buffer of all previous characters.
CLEARSCREEN Chap. 1	CS	Clears the graphics screens and returns Turtle to the home position.
CLEARTEXT Chap. 1	—	Clears the text screens.
CLOSEREAD Chap. 14	—	Closes a file you opened using OPENREAD.
CLOSEWRITE Chap. 14	—	Closes a file you opened using OPENWRITE.
COLUMN Chap. 14	—	Returns the current column position of the text screen cursor.
CONTENTS Chap. 13	—	Lists all global variables.
COPYOFF Chap. 4	—	Turns off COPYON.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
COPYON Chap. 4	—	Causes a dual output of all screen display to the printer.
COS Chap. 9	—	Calculates the cosine of a specified number.
COUNT Chap. 8	—	Returns the number of members in a word or list.
DATE Chap. 8	—	Returns the current date and time.
DO Chap. 10	—	Establishes a bottom-end control loop. Do repeats the execution of a procedure list as long as the subsequent WHILE function is TRUE.
DOT Chap. 5	—	Displays a dot on the graphics screen at specified coordinates.
EDIT Chap. 3	—	Causes LOGO to enter the edit mode.
ELSE Chap. 10	—	Runs a procedure list if the preceding IF test fails.
EMPTY? Chap. 8	—	Determines if an object has 0 members.
END Chap. 3	—	Indicates the conclusion of a procedure.

Primitive Chap. Ref.	Abbrev. If Any	Function
ERALL Chap. 4	—	Removes all current procedures from the workspace.
ERASE Chap. 4	—	Removes a specified procedure from the workspace.
ERASEFILE Chap. 4	—	Removes a specified file from a diskette.
ERROR Chap. 13	—	Simulates an error condition.
EXP Chap. 9	—	Raises a given number to the power of its exponent.
FENCE Chap. 13	—	Causes a procedure to halt and display an error message if the Turtle attempts to go beyond the graphics screen boundaries.
FILEPOS Chap. 14	—	Provides the current position of the file pointer.
FIRST Chap. 8	—	Extracts the first element of an object.
FIXED Chap. 9	—	Rounds a number to the next whole number of less value.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
FOR Chap. 1Ø	—	Performs an automatically increasing index for loop control.
FORWARD Chap. 1	FD	Moves Turtle forward a specified number of steps.
FPUT Chap. 8	—	Inserts an element at the front of an object.
FRACTION	—	Removes the integer portion of a number.
FULLSCREEN Chap. 2	—	Reserves the entire graphics screen for graphics and sets the number of graphics text lines to Ø.
GO Chap. 1Ø	—	Sends the execution of a procedure to the indicated LABEL position.
HEADING Chap. 5	—	Gives the Turtle's current heading.
HIDETURTLE Chap. 5	HT	Causes the image of the Turtle to disappear from the graphics screen.
HOME Chap. 1	—	Brings Turtle back to home position (the center of the screen pointed up).

Primitive Chap. Ref.	Abbrev. If Any	Function
IF Chap. 10	—	A condition test. IF conditionally executes a procedure list.
IFFALSE Chap. 10	—	Runs a procedure list if the condition register (set by TEST) contains FALSE.
IFTRUE Chap. 10	—	Runs a procedure list if the condition register (set by TEST) contains TRUE.
INTEGER Chap. 9	—	Removes the fractional portion of a fractional number.
ITEM Chap. 8	—	Extracts a specified element from a specified location in a word or list.
JOYX Chap. 12	—	Reads the X coordinate position of a specified joystick.
JOYY Chap. 12	—	Reads the Y coordinate position of a specified joystick.
KEY? Chap. 3	—	Determines if a key is pressed.
LABEL Chap. 10	—	Flags a position in a procedure for the GO function.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
LAST Chap. 8	—	Extracts the last element of an object.
LEFT Chap. 1	LT	Turns Turtle left a specified number of degrees in the range Ø to 36Ø.
LINE Chap. 14	—	Returns the current text line position.
LIST Chap. 8	—	Creates lists from other lists.
LIST?	—	Determines whether an object is a list.
LOAD Chap. 4	—	Copies a program or procedure from diskette to LOGO's workspace.
LOADPICT Chap. 4	—	Displays a picture file from diskette on the graphics screen.
LOCAL Chap. 7	—	Establishes a variable as a local.
LOG Chap. 9	—	Computes the natural log of a number.
LPUT Chap. 8	—	Inserts an element at the end of an object.
MAKE Chap. 7	—	Establishes variables and their values.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
MEMBER? Chap. 8	—	Determines whether a specified element is a member (is included in) an object.
MUSIC Chap. 6	—	Produces musical tones.
NOT Chap. 9	—	Performs a logical complement of an operation or word.
NOTRACE Chap. 13	—	Turns off LOGO's trace function.
NUMBER? Chap. 9	—	Determines whether a word is a number.
OPENREAD Chap. 14	—	Opens a file for reading data.
OPENWRITE Chap. 14	—	Opens a file for data input.
OUTPUT Chap. 7	—	Terminates the action of a procedure and returns its data to the calling procedure.
PADPENDOWN? Chap. 12	—	Determines if you press down the X-Pad pen.
PADX Chap. 12	—	Reads the X coordinate position of the X-Pad pen.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
PADY Chap. 12	—	Reads the Y coordinate position of the X-Pad pen.
PENCOLOR Chap. 2	PC	Displays the current pen color.
PENDOWN Chap. 5	PD	Enables the Turtle pen.
PENDOWN? Chap. 5	—	Returns the condition of the pen. Down = TRUE; UP = FALSE.
PENUP Chap. 5	PU	Disables the Turtle pen.
PIECE Chap. 8	—	Extracts a specified number of elements from a specified location in a word or list.
POALL Chap. 4	—	Displays the lines of all procedures currently in the workspace.
POTS Chap. 4	—	Displays the names of all procedures currently in the workspace.
PRECISION Chap. 9	—	Returns the current precision setting.
PRINT Chap. 14	—	Displays specified characters on the text screen and ends with an automatic carriage return.

Primitive Chap. Ref.	Abbrev. If Any	Function
PRINT1 Chap. 14	—	Prints specified text characters on the screen but does not produce a carriage return.
PRINTOUT Chap. 4	PO	Displays all lines of the specified procedure.
PRODUCT Chap. 9	—	Multiplies a series of numbers.
QUOTE Chap. 7	—	Defines a word, replaces quotation marks.
QUOTIENT Chap. 9	—	Divides 1 number by another.
RANDOM Chap. 9	—	Generates a random number.
RANDOMIZE Chap. 9	—	Generates a random initialization reference or seed to provide an alternate random sequence.
READ Chap. 14	—	Accesses data in a specified file.
READBYTE Chap. 14	—	Reads 1 byte of file data at a file's current position. The file position pointer automatically increases.
READCHARACTER Chap. 3	RC	Accepts 1-key input from the keyboard.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
REMAINDER Chap. 9	—	Determines the whole number remainder of a quotient.
REPEAT Chap. 1	—	Causes Turtle to repeat commands a specified number of times.
REQUEST Chap. 3	RQ	Accepts multiple-key input from the keyboard.
RERANDOM Chap. 9	—	Resets RANDOM to repeat the original random order.
RIGHT Chap. 1	RT	Turns Turtle right a specified number of degrees.
ROUND Chap. 9	—	Rounds a number that contains a fractional portion to the nearest whole number.
RUN Chap. 8	—	Executes a procedure name contained in a list.
SAVE Chap. 4	—	Copies a program or procedure to diskette.
SAVEPICT Chap. 4	—	Copies the contents of a graphics screen to diskette as a picture file.
SAY Chap. 11	—	Causes D.L. LOGO to say (speak) specified words or numbers.

Primitive Chap. Ref.	Abbrev. If Any	Function
SELECT Chap. 10	—	Executes the procedure list of the first subsequent TRUE condition.
SETBACKGROUND Chap. 2	SETBG	Sets the current background or screen color. There are 16 background colors.
SETCURSOR Chap. 14	—	Establishes the cursor at specified line and column position.
SETFILEPOS Chap. 14	—	Sets the file pointer to a specified position.
SETHEADING Chap. 5	—	Sets the heading of the Turtle in the range of 0-360 degrees.
SETPENCOLOR Chap. 2	SETPC	Establishes the current pen color. There are 4 pen colors for each background color.
SETPRECISION Chap. 9	—	Sets the precision of subsequent operations from 0 to 100 places.
SETSPLIT Chap. 2	—	Sets text lines in the graphics mode. SETSPLIT can be in the range 1-15.
SETX Chap. 5	—	Establishes the graphics screen X coordinate.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
SETXY Chap. 5	—	Establishes graphics screen X and Y coordinates.
SETY Chap. 5	—	Establishes the graphics screen Y coordinate.
SHELL Chap. 14	—	Lets you access OS-9 commands from within D.L. LOGO.
SHOWN? Chap. 5	—	Returns TRUE if the Turtle is visible or FALSE if it is hiding.
SHOWTURTLE Chap. 5	ST	Causes the image of the Turtle to reappear on the graphics screen.
SHUFFLE Chap. 9	—	Randomizes a list.
SIN Chap. 9	—	Calculates the sine of a specified number.
SOUND Chap. 11	—	Creates a sound from specified values representing pitch and duration.
SPLIT Chap. 2	—	Displays the current graphics screen split value.
SPLITSCREEN Chap. 2	—	Reserves text lines in the graphics screen mode.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
SQRT Chap. 9	—	Calculates the square root of a specified number.
SUM Chap. 9	—	Adds a series of numbers.
TAN Chap. 9	—	Calculates the tangent of a given number.
TEST Chap. 10	—	A condition test. Sets a condition register for subsequent IFTRUE or IFFALSE procedures.
TEXTSCREEN Chap. 2	—	Causes D.L. LOGO to return from the graphics mode to the immediate mode.
THING Chap. 7	—	Indicates that a word is a variable name. The same as a colon or dots.
TO Chap. 3	—	Specifies a procedure name.
TOWARDS Chap. 5	—	Provides the degree heading from Turtle's current position to a specified position.
TRACE Chap. 13	—	Turns on LOGO's trace function and displays procedure steps.

Primitive Chap. Ref.	**Abbrev.** If Any	**Function**
TURTLETEXT Chap. 5	—	Displays a message on the graphics screen at the current grid coordinates.
WHERE Chap. 8	—	Locates the position of an element in a word or list.
WHILE Chap. 10	—	Establishes a top-end control loop. WHILE executes a procedure *while* a specified condition is true.
WINDOW Chap. 5	—	Disables the WRAP condition. WINDOW displays only on-screen points.
WORD Chap. 8	—	Combines words to create 1 word.
WORD? Chap. 8	—	Determines whether an object is a word.
WRAP Chap. 5	—	Causes graphics to reappear on the opposite side of the screen when they go beyond a screen boundary.
WRITE Chap. 14	—	Transfers data to a specified file.

Primitive Chap. Ref.	Abbrev. If Any	Function
WRITEBYTE Chap. 14	—	Writes 1 byte of file data to the current file position. The file's position pointer automatically increases.
XCOR Chap. 5	—	Returns the current graphics screen X coordinate.
YCOR Chap. 5	—	Returns the current graphics screen Y coordinate.

APPENDIX D

Starting OS-9 from BASIC

If you do not have a Color Computer with BASIC version 1.1 or later or if you do not have the OS-9 System, you can type the following program and use it to start D.L. LOGO.

Enter this program from Disk Extended BASIC.

```
 10 REM ****************
 20 REM * BOOT OS-9 FROM BASIC
 30 REM ****************
 40 FOR I=0 TO 70
 50 READ A$
 60 POKE &H5000+I,VAL("&H"+A$)
 70 NEXT I
 80 CLS:PRINT "INSERT OS9 DISKETTE"
 90 PRINT "INTO DRIVE 0 AND PRESS A
    KEY"
100 A$=INKEY$:IF A$="" THEN 100
110 EXEC &H5000
120 DATA 86,22,8E,26,00,8D,0D
130 DATA FC,26,00,10,83,4F,53
140 DATA 26,03,7E,26,02,39,34
150 DATA 20,10,BE,C0,06,A7,22
```

```
160 DATA 86,02,A7,A4,6F,21,6F
170 DATA 23,6C,23,AF,24,10,BE
180 DATA C0,06,A6,23,81,13,27
190 DATA 12,AD,9F,C0,04,4D,27
200 DATA 06,6C,23,6C,24,20,E9
210 DATA 7F,FF,40,35,A0,4F,20
220 DATA F8
```

Type the following instruction at the `OK` prompt to save the above program:

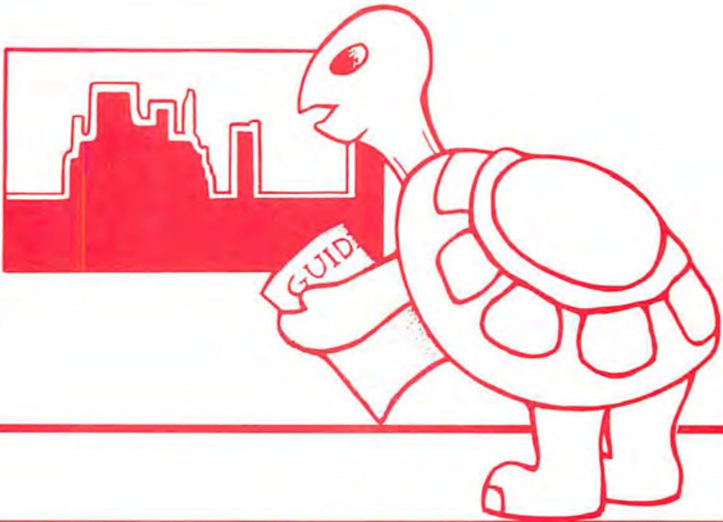
```
SAVE "*" ENTER
```

Type the following command to use this program to start D.L. LOGO:

```
RUN "*" ENTER
```

When the prompt appears, insert the D.L. LOGO diskette and answer the date and time prompts. When the OS-9 prompt appears, type:

```
LOGO ENTER
```



INDEX

Abbreviations, primitives 30
ABS 199, 212, 373
active keys 23
ADD
  procedure 217, 313
  program 302
adding
  items 166
  procedures 82
adults, and LOGO 8
AHEAD
  procedure 280 - 281, 282, 283
ALL, testing 226 - 227
ALLOF 201, 212, 226, 373
ampersand, music symbol 131
ANYOF 201 - 202, 212, 227, 373
APPEND 82 - 83, 88, 373
ARCS procedure 232
arguments 153
  definition 30
  in procedure names 149
arithmetic 186 - 219
  operators 188
  operations with variables 144 - 147
  powers 95
  syntax 190
arranging elements, SHUFFLE 208
  sorting 209 - 210
arrow keys 22 - 23
  editing 58 - 60
  left 24
  reverse 24
ASCII 192 - 193, 203, 212, 304, 374
  code 186
  data 312
ASK procedure 248 - 249
ATAN 200, 212, 374

BACK 26, 35, 374
BACKGROUND 48, 374
BACKUP, diskettes 4
background color 2, 40, 49
  chart 42
  repeating 43
backslash

Index

producing 24
special characters 150
BAD 'TO' STATEMENT, error 288
BAD PROCEDURE CALL, error 288
BALLOT program 105
BAND procedure 294
BAR procedure 98
bar graph program 360 - 361
bass clef, definition 113
beats, music 109
BG 43, 48
BI procedure 237
binary tree 237
BK 35
black and white television 3
Blow The Man Down score 113 - 114
BOMBER, sound program 355
BOMP procedures 252 - 253
BOMPA procedure 251
BOOKS program 182 - 184
boot procedures 5
BORDER procedure 294
bounds, graphics screen 31, 100
BOX procedure 55, 105, 144, 145, 155, 268
BOXDOWN procedure 233
boxes, drawing 43 - 44
BOXUP procedure 232
bracket symbols 24
BRICK procedure 293
BRICKS program 293
BULL procedure 147
program 364
BULL'S EYE program 364
BUMP program 228
BUTFIRST 163, 166, 179, 374
BUTLAST 163, 179, 374
BUTTIN procedure 265
BUTTON, joystick 265
BUTTON? 272, 374
BYE 35, 374
byte, reading and writing to
diskette 301 - 305

CALC program 150
calculating
CAN'T ERASE FILE, error 289
CAN'T OPEN FILE, error 289
CAT program 76, 348
CATALOG 76, 78, 87, 374
CATCH 291, 374
co-ordinates 93
calculations 186 - 219
in sound procedures 254 - 255
precision 205
use of spaces 190
with negative numbers 32
cartridge, speech/sound 246
catalog, program 348 - 350
commands, editor 331
change
directories 4, 75
text, procedures 60 - 61, 322
procedures 60 - 61, 65
screen modes 24
values 136
chapter summaries 35
CHAR 203 - 204, 213, 304, 374
CHARACTER procedure 193
characters
in procedure names 54
special Logo 150
values, ASCII 192 - 193
chart, pen and background
CHD 75, 87, 375
CHECK procedure 228, 243, 249, 308 - 309
checking Turtle, SHOWN? 99
children, and LOGO 8
CHOOSE procedure 218
choosing procedure names 56
chord progression 111

CIR procedure 238
CIR2 procedure 239
circle, drawing 50 - 51
CLEAN 45, 48, 375
CLEAR key 24
clearing keyboard buffer 66
CLEARINPUT, 68, 375
CLEARSCREEN 25, 29, 35, 45, 49, 100, 375
CLEARTEXT 29, 35, 375
clock, Logo time 177
CLOSEREAD 310, 375
CLOSEWRITE 310, 375
code lines, music 118
colon, variable names 139
color 3, 42
  settings 43
  invisible 40
colored ribbons program 359
COLUMN 311, 375
column, text screen 307 - 308
combining words 135 - 136
COMBINE procedure 183, 297 - 298
commands 26
  editing 70
  entering 5 - 7
  in procedures 63
  semicolon 69
common errors 283
comparative operations 191
COMPARE procedure 224
comparisons 201 - 204
  using IF 223 - 224
  using SELECT 225
compass headings 96
complete variable display 140
COMPOSE program 219
computer languages 8
containers, variables 133 - 155
CONTENTS 149, 278, 291, 375
contents, variable 141
control
  editor 328
  key 23
coordinates 30, 92 - 94
  system 31
copy diskettes 4
COPYOFF 88, 375
COPYON 88, 376
CORNER procedure 105
correcting errors 276
COS 199, 212, 376
COUNT 168, 180, 376
COUNTDOWN procedure 164 - 165
counting members in an object 168
CREATE procedure 296
creating 36 - 37
  errors 283
CROSS procedure 105
CS 35, 45, 49
CTRL, keys
cursor, moving with arrow keys 22 - 23
cycles per second, sound 250

data list, variables 136
DATA procedure 313
  program 313 - 317
DATE 180, 376
date 177
  time 5
debugging aid
  CONTENTS 278
  PAUSE 281
  TRACE 285
debugging programs 275 - 294
decrement, variable values 145
degrees 33
  heading 104
delete
  files 66
  files, PURGE program 350 - 351
  text 321, 329 - 330

elements from an object 171 - 172
DELETE procedure 316
DEMO program 151
demonstration program 6 - 7
descriptions, error codes 287 - 290
DIAG procedure 308
Disk BASIC 4
DISK ERROR 289
DISK FULL error 289
DISPLAY procedure 218, 316
directions 26, 27,
headings 95
directories, change 4, changing 75
disk directories 76
diskette 88
programs 6
picture files 77
saving procedures 74
displaying
a procedure 84
complete variable 140
special characters 150
DIV procedure 218
DIVIDE BY ZERO, error 287
DO 230, 376
DONUT program 365
DOREME procedure 110
dots, variable names 134
DOT 97, 103, 376
DRAW procedure, X-Pad 267
drawing a circle 50 - 51
DSKINI 4

EDIT 56, 68, 376
Editor, cursor control 328
edge, graphic screen 100
edit 57 - 63
changing text 322
copying text 323
cursor 58 - 60
deleting text 321
features, summary 320
files 328
functions 319 - 334
inserting text 321
keys 70
mode 54, 55, 57
moving text 324
options 57
replacing text 326
searching text 326
using arrow keys 58 - 60
editor
change commands 331
delete commands 329 - 330
insert commands 329
search and replace 332
special commands 333
yank command 332
education 8
elements
inserting 173 - 174
of an object 170
ELSE 225, 240, 376
EMPTY OBJECT,
EMPTY? 165, 180, 299, 376
END 68, 376
procedure 57
end
of a word 134
procedure line 69
ENDVIEW procedure 184
ENTER,
ending lines 62
equipment
minimum 3
optimal 3
ERALL 81 - 83, 87, 377
ERASE 81 - 82, 87, 377
ERASEFILE 87, 377
erasing

files from diskette 78 - 79
files in the workspace 80 - 81
ERROR 283, 291, 377
errors 275 - 294
codes and messages 287 - 290
messages 28, 287 - 291
syntax 25
typing 277
user defined 290
EXAM procedure 315 - 316
execute
a primitive 22
procedure lines 175 - 176
procedures 55 - 56
exit
D.L. LOGO 34, 306
insert mode 59 - 60
the edit mode 57
EXP 198 - 199, 212, 377
external
devices 260
Turtle 2
extracting items 169

FACT procedure 153
FALL, sound procedure 355
false operations 191
FD 35
FENCE 100, 103, 291, 377
fields, definition 299
filename 6 - 7, 88
FILE NOT OPEN, error 289
file pointer index 302
FILEPOS 301, 310, 377
files 296
deleting 66
erasing from diskette 78 - 79
erasing from workspace 80 - 81
opening 296 - 297
reading bytes 304
reading data 298 - 300
writing to 298 - 300
filling a circle 50 - 51
finding
errors 276
items 167
FINISH procedure 244
FIRST 163, 166, 179, 377
FIXED 197 - 198, 211, 377
FLAKE program 237
flat, music 131
FLOWER procedure 71 - 72, 81
FLY procedure 255
FOR, loops 232 - 233, 378
foreground colors 2
format
music 118
procedure 57
text screen 307 - 309
FORWARD 25, 35, 378
FPUT 163, 179, 378
FRACTION 198, 211, 378
FRAME procedure 80
FULLSCREEN 24, 45, 48, 378

geography 92
GET procedure 264
GETFILE procedure 317
GETSET procedure 309
GIVE procedure 150 - 151
global
variable 148 - 149, 153
search and replace 119 - 120
GO 177, 233 - 235, 378
GRAPH program 360 - 361
graphics
definition 31
display 31
hiding 46
language 2

mode 2
reproducing on a printer 77
saving to diskette 77
setting points 97
screen bounds 31
screen grid 92 -94
using the full screen 45
GRAPHICS program, swirl 351
greater-than operations 191
grid system 31, 92 - 94
GUESS program 365 - 371

harmony, music 122 - 123
HEADING 95, 102, 378
heading
increments 104
procedure 57
hiding graphics 46
HIDETURTLE 10, 98, 378
HIGHER program 149
HOME 27, 35, 100, 378
HT 103, 378

IF 222 - 223, 379
IFFALSE 224, 379
IFTRUE 224, 379
immediate mode 55
indention in procedures 57
index, file pointer 302
INPUT procedure 182 - 183, 194, 242 - 243, 297, 313 - 314
input 297
to procedures 144 - 145, 150 - 154
variables 151
joysticks 260 - 265
keyboard 64 - 66
keyboard 66
insert commands 329
INSERT program 173 - 174
inserting
elements in an object 173 - 174
in procedures 59 - 60
in text, editing 321
ISOLATED OBJECT, error 288
ITEM 163, 167 - 168, 179
INTEGER 197, 211, 379
interfacing 2
invisible Turtle 98
invisible line 40
ITEM 167 - 168, 379

JOY procedure 261, 264, 281 - 282
JOY2 procedure 261
JOY3 procedure 262
joystick 2
output range 272
reference 272
BUTTON primitive 265, 374
joysticks 260 - 265
JOYX 272, 379
JOYY 272, 379

KEY? 66 - 68, 379
key signature 116
keyboard
buffer, clearing 66
entries 22
input 64 - 66
repeating keys 23
special keys 22 - 24
keypress, wait for 66 - 67
keys
for editing 70
CLEAR 24
CTRL 24
active 23
typing 6
KILL procedure 66 immediate mode 55

LABEL 233 - 235, 379
languages 9
LAST 163, 166, 179, 380
LAZER procedure 254
LEFT 26, 35, 380
left arrow 24
legal characters in procedure names 54
length of objects 168
less-than operations 191
let, see MAKE 134
library 180
    of procedures 82
LINE 311, 380
line, text screen 307 - 308
lines, setting text 46 - 47
link procedures 54
LINK program 141 - 142
linking
    variables 140 - 141
    storing data 141 - 142
LIST 161, 179 - 180, 380
LIST? 179, 380
list 180
    variable 137
listing a procedure 84
lists
    manipulation 163 - 174
    operating on 160
    variables 134 - 135
    within lists 138 - 139
LOAD 87, 380
LOADPICT 87, 380
LOCAL 153, 380
local variable 148 - 149, 153
locations on the screen 92
LOG 198, 211, 380
LOG OF NEG NUMBER, error 289
logic operators 189
logical concepts 2
LOGO
    letters, graphics program 351 - 353
    starting 5
    to exit 34
long lines 62
loop operations 222
loops 230 - 244
    using FOR 232
    logic testing 222
    nested 231 - 232
    recursion 236 - 239
lowercase characters 23
LPUT 163, 166 - 167, 179, 380
LT 35, 380

MAKE 134, 153, 380
make, variables 134
manipulate
    procedures 60
    words, lists, sentences 163 - 174
math, arithmetic operators 188
MATH program 216 - 217
mechanical device 2
MEMBER? 163, 180, 381
memory 88
MEMORY FULL, error 288
MENU procedure 182, 216
MENU program 175 - 176
MINE procedure 243
minutes 178
MISSING ')', error 288
MISSING ']', error 288
MISSING ARGUMENT, error 287
mistake
    typing 23
    correcting 26
modes, definition
move cursor, arrow key 22
MOVE procedure 64, 230, 231
MOVEMORE procedure 65
moving text 324
MULT procedure 217

Multi-Pak 3
music 107 - 132, 381
  editing 131
  worksheet 132
  score 108
MUST BE A LIST, error 287
MUST BE A NUMBER, error 287
MUST BE A PROC LIST, error 288
MUST BE A WORD, error 287

NEXT procedure 294
NOISES, sound programs 355
NOT 202 - 203, 212, 228 - 229, 381
NOT FOUND, error 288
NOT TRUE/FALSE, error 287
NOTRACE 291, 381
NUMBER 204, 213
NUMBER OUT OF RANGE, error 287
name, variable 141
names
  choosing procedure 56
  procedure 69
  variable 134
neat, making procedures 61 - 63
negative numbers 32
nest procedures 54
nests 231
note, definition 112
note-word 109
notes, LOGO 2
number of inputs to procedures 151
numbers 181, 186 - 219
  in lists and words 190

object
  definition 153
  deleting elements 171 - 172
  finding elements in 170
  input to procedures 184
OBJECT procedure 219
OCT program 357
octagon program 357
octave 132
  switches 131
  definition 113
  music 108 - 109
OOPS procedure 281
opening files 296
OPENREAD 310, 381
OPENWRITE 296, 310, 381
OPERATION procedure 218
order in the workspace 57
OS-9 2, 4
  boot 5
  commands 4
  format 88
  functions 306
  starting from BASIC 391 - 392
out of bounds error 281
OUTPUT 153, 381
output
  from procedures 150 - 154
  recursive 151 - 152

PAD program 268 - 269
PADPENDOWN? 266
PADPENDOWN? 272
PADX 266, 272, 381
PADY 266, 272, 382
parent procedure 149
parentheses 179
PATH program, wandering Turtle 356
PATHS program 353
PAUSE 281
pause, music 118 - 119
PAWS procedure 282, 284
PC 43, 48
PD 103

PENCOLOR 48, 382
pen color 40, 49
changing 96 - 97
chart 42
default 49
repeating 43
pen, X-Pad 266
pen control, X-Pad 266 - 267
PENDOWN 96 - 97, 103, 382
PENDOWN? 103, 382
PENUP 96 - 97, 103, 382
PETAL procedure 72
philosophy of LOGO 8
PHONE procedure 254
phonetic spelling 257
PICK procedure 313
picture files 77
pie chart program 363 - 364
PIE program 363 - 364
PIECE 163, 169, 180, 382
pitch 108
selections, music 131
definition 113
sound 250 - 251
PLAY procedure 242, 274
plotter 2
PO 84, 88, 383
POALL 84, 382
position of Turtle 93, 104
POSITION procedure 267
POTS 84, 88, 382
power symbol, arithmetic 23, 95
precision 186, calculations 205
PRECISION 213, 382
primitive
abbreviations 30
and procedure names 59
PRINT 68, 311, 382
PRINT1 64 - 65, 68, 311, 383
printer 3, 84 - 85, 88
graphic reproduction 77
procedures 84 - 85
PRINTOUT 84, 88, 383
PROBLEM procedure 217
procedures 7, 28, 54 - 56
accepting input 144 - 145
adding comments 63
appending 82 - 83
format 57
indentation 57
input 150 - 154
making neat 61 - 63
number of inputs 151
library 82
line ending 69
lines, inserting 59 - 60
lines, numbering 59
listing 84
names 54, 56, 69
names with arguments 149
names, spaces 55
output 150 - 154
saving 74 - 75
spacing between 59
storage, workspace 60
style 57
PRODUCT 195, 211, 383
program 7
building 56
line execution 175 - 176
lines 62
programming for the X-Pad 266
programs 54
diskette 6
producing a cube 36 - 37
sample 348 - 371
prompt, question mark 22 PENCOLOR 48
PURGE program 350 - 351
PUT procedure 317

QUE procedure 264

question mark, prompt 22
QUIT procedure 184, 317
QUIZ procedure 192, 208, 263 - 265, 308 - 309
QUOTE 135, 153, 383
quotation marks 181
  and variables 134
  using 172
QUOTIENT 195, 211, 383

RAD procedure 95 - 96
RANDOM 177 - 178, 206 - 207, 213, 383
random 206 - 207
RANDOMIZE 207, 213, 383
RANGE procedure 251
RC 64, 68, 383
READ 310, 383
READANS procedure 264
READBYTE 301, 304, 310, 383
READCHARACTER 64, 68, 383
re-enter D.L. LOGO 306
reading files 298 - 300
reading the joystick 260 - 265
READIT procedure 64
rearranging
  SHUFFLE 208
  sorting 209 - 210
REC procedure 84
records, definition 298
RECTANGLE procedure 57 - 58, 234
recursion 151, 236 - 239
recursive outputs 151 - 152
references, error codes and messages 287 - 290
repeating
  colors 43
  keyboard entries 23
recursion 151
REMAINDER 196, 211, 384
REMOVE program 171 - 172
REPEAT 32, 35, 230, 384
replace 119 - 120
  text 60, 326
reproducing graphics on a printer 77
REQUEST 65, 68, 384
RERANDOM 207, 213, 384
restricting Turtle movement 100 - 101
REVER program 354
reverse a word program 354
reverse arrow 24
reversing test results, NOT 228 - 229
rhythm 108
RIBBON program 359
RIGHT 26, 35, 384
ring of flowers program 358
RING procedure 278
rotate procedure 56
root program 356
ROOTS, arithmetic
ROUND 196 - 197, 211, 384
ROW procedure 293
RQ 65, 68, 194, 384
RT 35, 384
RUN 175, 180, 384

sample programs, Appendix B 348 - 371
SAVE 75, 87, 384
SAVEPICT 77, 87, 384
saving
  files 296
  multiple procedures 75
  procedures 74 - 75
SAY 246 - 249, 257, 384
screen
  colors 40
  directions 27
  limits 100 - 101
  locations 92
  mode, changing 24
SEE procedure 303

SEEK program 242 - 244
SELCT procedure 315
SELECT 192, 225, 240, 385
SEND procedure 86
SENTENCE 139, 162
SET procedure 293
SETBACKGROUND 43, 48, 385
SETBG 42, 48, 385
SETCURSOR 311, 385
SETDOT procedure 243
SETFILEPOS 301, 310, 385
SETHEADING 95, 102, 385
SETPC 40, 43, 48, 385
SETPENCOLOR 48, 385
SETPRECISION 213, 385
SETSPLIT 49, 385
SETUP procedure 249
SETX 102, 385
SETXY 92 - 94, 102, 386
SETY 102, 386
Seymour Papert 8
SHAPE program 233 - 235
SHELL 306, 311, 386
SHOW 137
    implied 161
SHOWFILE procedure 299
SHOWN? 99, 103, 386
SHOWQUES procedure 264
SHOWTURTLE 98, 103, 386
SHUFFLE 208 - 209, 213, 386
SIN 199 - 200, 212, 386
SINE procedure 216
size 25
screen
    graphics grid 92 - 94
    setting graphics 46
    setting points 97
    text 307
    selecting 46 - 47
search 119 - 120
search and replace, editor 332
search text 326
searching for elements in an object 170 - 171
seconds 178
selecting pieces of an object 169
semicolon 69
sentence 180
    manipulation 163 - 174
    words, lists 163
setting
    color 43
    screen lines 46
    screen points 97
sharp, music 131
shortcut, note entry in music 130
single command mode 5
size of screen 25
sorting 209 - 210
SOUND 250, 257, 386
sounds, unusual 256
SPACE procedure 254
spaces
    between commands 36
    between primitives 36
    between procedures 59
    in calculations 190
    in procedure names 55
    in words 164
special commands, editor 333
speech 246 - 249
    cartridge 246
speed, increasing 104
SPELL program 248 - 249
spelling
    program 248 - 249
    phonetic 257
spider's web program 359
SPIN 60,
SPIN procedure 325
spin procedure 58 - 60, 325
SPIN2 procedure 82
SPIRAL

procedure 146
program 360
SPLIT 46 - 47, 49, 386
split procedure lines 60
SPLITSCREEN 24, 46, 49, 386
SPOKE procedure 277 - 278
square bracket, producing 23, 36
SQRT 200, 212, 387
SQRT OF NEG NUMBER, error 289
SQUARE procedure 234
ST 103, 386
STACK program 155
staff, definition 112
starting D.L. LOGO 5
STEM procedure 81
steps, definition 25
stop execution 22
storing
data, linking 141 - 142
procedures on diskette 75
SUB procedure 217
SUBJECT procedure 219
SUM 194, 211, 387
SUN procedure 96
SWIRL
procedure 63
program 351
symbol, arithmetic power 23
syntax 25 ST 103
SYNTAX ERROR, error 288

talk 246 - 249
TALK procedure 247
TAN 200, 212, 387
TATTLE procedure 93
teleporting 93
TELLANS procedure 265
template, X-Pad 268 - 269
tempo settings, music 131
TEST 186, 223 - 224, 240, 387
testing
logic operators 189
logical tests in loops 222
using ALLOF 227
using ANYOF 227
text
change 60, 322, 331
copying with yank 323
display 31
delete commands 321, 329 - 330
editor cursor control 328
files 6
handling 296 - 305
insert commands 329
inserting 321
moving 324
replace 60
replacing 326
screen 307, formatting 307 - 309
search and replace 332
searching 326
special editing commands 333
yank command 332
TEXTSCREEN 25, 49, 387
T&F procedure 265
THING 139, 153, 387
THUMP, sound program 355
time 177
date 5
music 115
TIME procedure 177
TIMELOOP procedure 178
TO 55 - 56, 68, 387
TOWARDS 95, 102, 387
TRACE 285, 291, 387
trace to printer 291
treble clef, definition 113
tree, graphic 237
TREE procedure 237
TRIANGLE procedure 235
triangular ring program 357

TRIRING program 357
true operations 191
turtle shape 27
twisting triangle, program 353
TURTLE OUT OF BOUNDS, error 289
TURTLETEXT 94, 102, 388
Turtle
    position 93
    writing 93 - 94
    checking status 99
    hiding 98
types of variables 136
typing errors 23, 277

UNDEFINED SYMBOL, error 288
UNDEFINED PROCEDURE, error 288
unusual sounds 256
uppercase characters 23
user defined errors 290

value,
VARI procedure 149
variable 141
values, variable 136
variable contents 139, 141
    display, CONTENTS 278
    decrement of
    displaying completely 140
    global 148 - 149
    in arithmetic operations 144 - 147, 191
    linking 140 - 141
    local 148 - 149
    making 134 - 135
    name 134, 141, 153
    sentence 139
    types 136
    value 141, losing 291
VERB procedure 219
view diskette directories 76
VIEW procedure 183
vocabulary, procedure names 59
voices, music 131

wait for keypress 66 - 67
wandering Turtle program 356
WEB program 359
WEAVER procedure 101
WHERE 180, 388
WHILE 230 - 231, 240, 388
WINDOW 100 - 101, 104, 388
WORD 161, 179, 388
WORD TOO LONG, error 289
WORD? 179, 388
words 180
    combining 135 - 136
    manipulation 163 - 174
    operating on 160
    variables 134 - 135
worksheet, music 132
workspace
    computer memory 60
    erasing files 80 - 81
    order 57
    saving to diskette 75
WRAP 100 - 101, 103, 388
WREATH program 358
WRITE 310, 388
WRITEBYTE 301, 304, 311, 389
writing
    to files 298 - 300
    Turtle text 93

X character, music 131
X-Pad 2, 3, 266
    grid 266
    output range 272
XCOR 93, 102, 389

Yank, editor 332
yanking text, copying 323 YCOR 93
YCOR 93, 102, 389

ZAP procedure 254

RADIO SHACK, A Division of Tandy Corporation

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

AUSTRALIA	BELGIUM	FRANCE	U. K.
91 Kurrajong Avenue Mount Druitt. N.S.W. 2770	Rue des Pieds d'Alouette, 39 5140 Naninne (Namur)	BP 147-95022 Cergy Pontoise Cedex	Bilston Road Wednesbury West Midlands WS10 7JN

D.L. Logo is fun. D.L. Logo is a computer language designed for enjoyment. It lets you weave shapes, colors, relationships, speech, music, and sound into infinite patterns. Easy-to-learn "English" commands let you explore fascinating new concepts at multiple levels. Use the sample programs in this manual to create dazzling graphics, teach spelling and arthmetic, play games, and help you manage your diskettes.

D.L. Logo is challenging. Logo's powerful arithmetic, mathematic, trigonometric, and Boolean functions will stretch your logic and conceptual abilities to the limit. Transform geometric concepts into kaleidoscopic visual displays, doodle, explore. Even a mistake you make in logic might introduce an intriguing adventure, or possibly a new logic.

D.L. Logo is for you. Whether you are an accomplished programmer, a beginner, or anywhere between, D.L. Logo can help you strengthen your programming skills. It can educate and entertain, providing you and your friends with hours of enjoyment. With D.L. Logo, machine language programmers can provide interface to external devices such as robots, a mechanical turtle, or a plotter.

To begin your adventures with D.L. Logo, you need:

- A Tandy Color Computer with 64K memory
- At least one disk drive
- A television (color recommended)

To make full use of D.L. Logo's potential, you also need:

- One or two joysticks
- The Multi-Pak Interface
- An X-Pad
- The Speech/Sound Cartridge

10/85-TP **874-9790** **Printed in U.S.A.**

READ ME FIRST

All computer software is subject to change, correction, or improvement as the manufacturer receives customer comments and experiences. Radio Shack has established a system to keep you immediately informed of any reported problems with this software, and the solutions. We have a customer service network including representatives in many Radio Shack Computer Centers, and a large group in Fort Worth, Texas, to help with any specific errors you may find in your use of the programs. We will also furnish information on any improvements or changes that are "cut in" on later production versions.

To take advantage of these services, you must do three things:

(1) Send in the postage-paid software registration card included in this manual immediately. (Postage must be affixed in Canada.)

(2) If you change your address, you must send us a change of address card (enclosed), listing your old address exactly as it is currently on file with us.

(3) As we furnish updates or "patches", and you update your software, you must keep an accurate record of the current version numbers on the logs below. (The version number will be furnished with each update.)

Keep this card in your manual at all times, and refer to the current version numbers when requesting information or help from us. Thank you.

APPLICATIONS SOFTWARE VERSION LOG			OP. SYSTEM VERSION LOG
Ø1.ØØ.ØØ			

Read Carefully

In order for us to notify you of modifications or updates to this program you MUST complete this card and return it immediately. This card gets you information only and is NOT a warranty registration. Register one software package per card only. The registration card is postage paid—it costs you nothing to mail.

Two change of address cards have been included so that you may continue to receive information in the event that you move. Copy all address information from the Registration Card onto them prior to sending the Registration Card. They must show your "old address" exactly as you originally registered it with us.

Software Registration Card

Cat. No. 2603033

Version 01.00.00

Name ____________________

Company ____________________

Address ____________________

City ____________ Phone (   ) ____ - ____

State ____________ Zip ____________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 138 FORT WORTH, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

Software Registration
Data Processing Dept.
P.O. Box 2910
Fort Worth, Texas 76113-9965

TANDY

Color Computer
64K

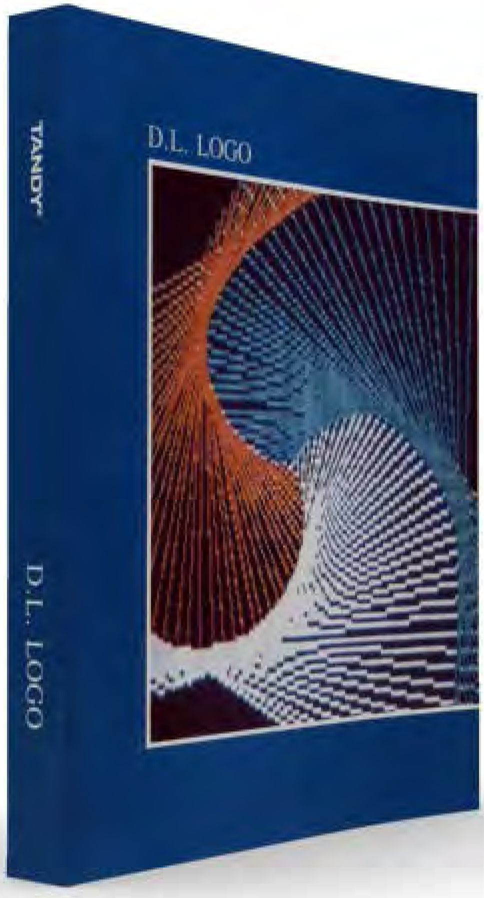
Cat. No. 26-3033

DL LOGO (With OS-9 Runtime)

DALE LEAR 1985. ALL RIGHTS RESERVED. LICENSED TO TANDY CORPORATION

01.00.00

TANDY®


D.L. LOGO
TANDY
D.L. LOGO

D.L. LOGO
For the OS-9
Operating System
D.L. LOGO

TANDY
Color Computer
64K
DL LOGO (With OS-9 Runtime)
01.00.00