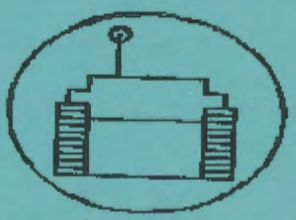
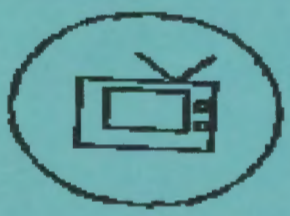
Journal
of the ISTE
Special Interest Group
for Logo-Using
Educators

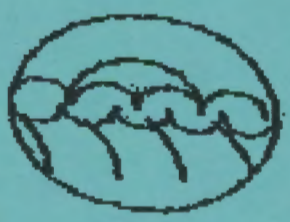
LOGO EXCHANGE

Spring 1994 Volume 12 Number 3

Graphics in This Issue:
LogoPOGs







Also—
Logo Procedures for Paralyzed Children
Using* LogoWriter *to Tune Turtles and Make Sweet Music
Hypermedia Design—Dynamic Information Management

International Society for Technology in Education

LX Volume 12 Number 3

Submission of Manuscripts

Logo Exchange is published quarterly by the International Society for Technology in Education Special Interest Group for Logo-Using Educators. *Logo Exchange* solicits articles on all aspects of Logo use in education. Articles appropriate to the International column should be submitted directly to Dennis Harper. Advanced articles should be submitted to Mark Horney, editor of the Extra for Experts column. Articles appropriate for the MathWorlds column should be sent directly to Sandy Dawson.

Manuscripts should be sent by surface mail on a 3.5" disk (where possible). Preferred format is Microsoft *Word* for the Macintosh. ASCII files in either Macintosh or DOS format are also welcome. Submissions may be made by electronic mail as well. Where possible, graphics should also be submitted electronically. Please include electronic copy, either on disk (preferred) or by electronic mail, with any paper submissions. Paper submissions alone will NOT be accepted.

Send surface mail to:

Sharon Yoder
170 Education, DLIL
University of Oregon
Eugene, OR 97403

Send electronic mail to:

Internet: YODER@oregon.uoregon.edu

Deadlines

To be considered for publication, manuscripts must be received by the dates indicated below.

Volume 13, Number 2	Apr. 1, 1994
Volume 13, Number 3	July 1, 1994
Volume 13, Number 4	Oct. 1, 1994
Volume 14, Number 1	Feb. 1, 1995

Logo Exchange is published quarterly by the International Society for Technology in Education (ISTE), 1787 Agate Street, Eugene, OR 97403-1923, USA; 800/336-5191. This publication was produced using *Aldus PageMaker®*.

Individual ISTE Members may join SIG/Logo for $20.00. Dues include a subscription to *Logo Exchange*. Add $10 for mailing outside the USA. Send membership dues to ISTE. Add $2.50 for processing if payment does not accompany your dues. VISA, Mastercard, and Discover accepted.

Advertising space in *Logo Exchange* is limited. Please contact ISTE's advertising coordinator for space availability and details.

Logo Exchange solicits articles on all topics of interest to Logo-using educators. Submission guidelines can be obtained by contacting the editor. Opinions expressed in this publication are those of the authors and do not necessarily represent or reflect the official policy of ISTE.

POSTMASTER: Send address changes to *Logo Exchange*, ISTE, 1787 Agate St., Eugene, OR 97403-1923. Second-class postage paid at Eugene OR. USPS# 660-130. ISTE is a nonprofit organization with its main offices housed at the University of Oregon. ISSN# 0888-6970

LOGO EXCHANGE

Volume 12 Number 3 Journal of the ISTE Special Interest Group for Logo-Using Educators Spring 1994

Contents

From the Guest Editor

Logo—A Garden for the Mind

by Dorothy Fitch

Gardening is an individual process. People plan their gardens in different ways and grow different types of plants. A garden takes on the personality of its gardener.

Logo is a garden that you can design and enjoy. The language provides you with many built-in tools to get you started—the equivalent of a shovel, trowel, rake, watering can, peat moss, and fertilizer. You provide the land. You determine how big your garden will be and how it will be designed. You also create your garden's personality and unique character.

At first you might be comfortable tending just a small plot and growing some traditional flowers. As you gain confidence and your flowers come up the way you want them to, you might expand your garden area and the type of gardens you grow.

A **vegetable** (or herb) **garden** is functional and utilitarian. You grow things for a purpose. You will use its produce in many different ways. In a Logo vegetable garden, you might grow a set of tools that will come in handy later. They supplement the tools Logo has already given you.

Under a bird feeder grows a **wildflower garden.** Sunflowers, thistles, daisies, and buttercups thrive in random places, without any planning from you. It's a random Logo garden—a "let's see what happens" garden.

In contrast, a **formal garden** is carefully planned. You don't just start planting a formal garden. Before you even pick up a garden tool, you must lay out the design for the beds and decide where your plantings will go. Professional gardeners always plan their designs carefully, using gardening guidelines and their own creativity. You can design structured gardens in Logo. Even though a garden is structured, it doesn't have to be boring. It might include a whimsical **topiary garden,** with bushes sculpted into animals, or a **maze** of hedges!

A **hummingbird garden** or a **butterfly garden** is a visually appealing collection of colorful flowers. Logo gardens are often designed for their visual appeal, to attract attention.

Some Logo gardeners work on their garden for a while, and then stand back to see how it looks. They may rearrange a corner or turn an object or replace one item with another until it "feels" right. To a designer of a **Japanese garden,** aesthetics are all-important. You may not know in advance which plant or rock should go where, but when it's in the right place, you know it. The garden may be planned more from the middle-out, as Logo projects often are. There is a plan, but it is adaptable along the way.

There are many other types of gardens that could invite further analogies: annual gardens, perennial gardens, rose gardens, country gardens, botanical gardens, arboretums, tropical gardens, and so on. I'll leave these to you to ponder.

Of course, every garden has its share of obstacles. There will be **bugs.** You learn to deal with bugs, and, as you gain more experience, control or prevent them. Your garden may also have rocks—rocks that seem to grow up over the winter into your garden, all by themselves. These will challenge you. You will have to figure out how to approach these problems. Once you have solved them, you're free to get back to the gardening at hand.

With Logo, you can plant a different type of garden each day. You're not restricted to a style, a design, or an approach. Logo provides you with an environment in which you can grow and cultivate whatever you want, however you choose.

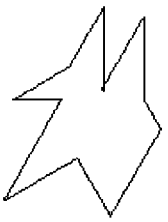
Your Classroom as an Environment, Your Students as the Flowers

Real gardens and Logo gardens grow in different types of environments. Your classroom may have limited resources, but just as you can grow flowers in containers on rooftops, you can use Logo in a small classroom with just one computer. How well your flowers grow depends less on the space than on the quality of care and nourishment. Flowers can thrive with the right encouragement, even if the garden seems small and meager.

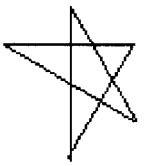
You are always planting seeds with Logo. Some seeds take a while to germinate and become visible. You will plant seeds and not even know it. One day, you may see something sprout from one of these seeds. Here are two examples of children's work, done over a period of time: in one case, just a day; in the other case, an entire year. You may recall these children's projects from one of my previous *LX* columns. I thought they were worth sharing again in this context.

Cathy's Stars

Using *Kinderlogo*, a single-keystroke program, Cathy (age 6) drew this star one morning. Her classroom had just one computer, so her time with Logo was limited. When her time was up, she asked if her star design could be saved, although she didn't appear quite satisfied with it.

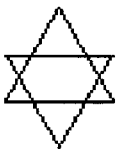


Later that day, the teacher felt a tug on her arm. It was Cathy, asking if she could have another turn at the computer. Normally, only one turn was allowed, but her teacher felt that this request was special. Cathy drew another star, using a different approach. She was much more pleased with this second star.



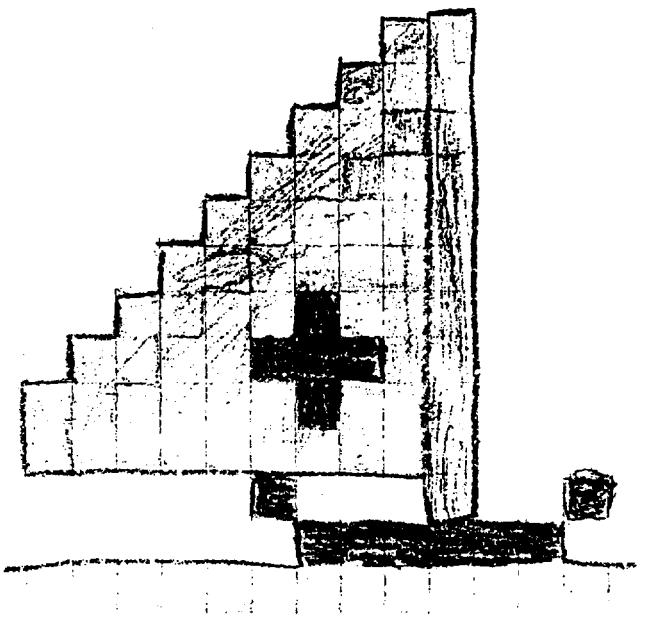
Had Cathy spent that day drawing stars with a pencil and thinking about how to draw a more "star-like" star? We don't know. But a seed was planted in the morning that germinated into a new idea sometime during the course of the day, right in the middle of regular classroom activities. Logo sometimes gives us an opportunity to see this happen. It opens up a window to the mind.

Later in the year, Cathy drew this design, an even more balanced and refined star. Is hers a perennial garden of stars?

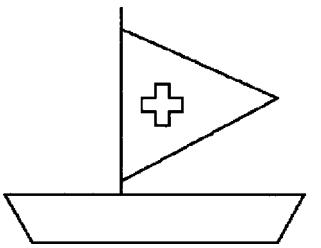


Freddie's Boats

Freddie, a first-grader, also used *Kinderlogo*. At the beginning of the year, he did a graph-paper activity designed to help him get familiar with the two-dimensional computer screen. He drew this boat design, which his teacher kept in a folder along with the work of other students.



At the end of the year, Freddie drew this boat using single-keystroke *Kinderlogo* commands.



You can imagine his teacher's surprise when she discovered his graph paper design in June! She had forgotten about his boat, but obviously he hadn't. This seed of an idea lived quietly all year, eventually reappearing in a Logo design. The design is almost the same, though the boat is headed in a different direction. Perhaps the wind changed.

After Logo, What Next?

Most Logo gardeners are content to cultivate small areas of land. But some people who have spent a long time gardening with Logo and become very productive may want to go commercial and have a farm. Right now, most versions of Logo do not offer industrial-strength tools like combines and tractors. But who knows what the future might hold?

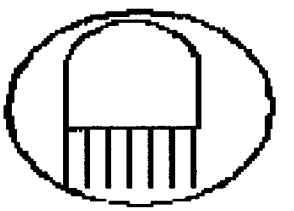
Through Logo, you can teach basic gardening concepts and skills—how to work the soil, how to handle different types of plants, and how to deal with bugs and rocky patches. Making the transition to a career of farming is not difficult because the foundation has already been built.

Whether or not your students pursue a career in gardening, spending time with Logo gardens opens up their minds to new ideas, gives them choices, and teaches them to overcome obstacles. We may never know to what extent Logo plants seeds in their minds that blossom long after we pass out of their lives.

Dorothy Fitch has been director of product development at Terrapin since 1987. A former music educator, she has also directed a computer education classroom for teachers and students and provided inservice training and curriculum development for schools. She is the author of *Logo Data Toolkit* and coauthor of *Kinderlogo*, a single-keystroke Logo curriculum for young learners. At Terrapin, she coordinates software development, edits curriculum materials, writes documentation, and presents sessions at regional and national conferences.

Dorothy Fitch
Terrapin Software, Inc.
400 Riverside Street
Portland, ME 04103-1068

CompuServe: 71760,366
Internet: 71760.366@compuserve.com



by Barret Mejia and
Nigel Anguita

The Little Toolmaker: "Dad, can you help me?"

by Tom Lough

The words were music to this father's ears. I always look forward to doing things with Kyser, our 10-year-old. "Of course, son. What is it?"

He explained that there had been a small toy in his fast-food kid's meal, and he was curious about how the various parts moved when he pushed it back and forth across the table. As is true of untold legions of children stretching back into the depths of time, he wanted to take the toy apart to see how it worked.

Evidently the toy designer had anticipated this curiosity. Although the parts of the toy were held together with small screws, they did not have slots for a flat tip screwdriver—or even an "x" for a Phillips screwdriver.

Instead, each small screw head had a triangular-shaped recess. Hmmph! There were not many places I knew where a 10-year-old (or the father of a 10-year-old, for that matter) could purchase a screwdriver with a triangular-shaped tip.

For a few minutes, we both sort of fussed around out of general frustration. Then I got an idea. "Why not make a triangular screwdriver?"

"How are we going to do that, Dad?"

I explained that maybe we could file the end of a small metal rod into the shape of a triangle. Down in the basement, we found a short copper rod scrap and secured it in an old vise given to me by my grandfather. Talking about Granddaddy as we prepared to work gave the activity a very special mood.

Next, we grabbed a file and took turns pushing and pulling it back and forth along the side of the rod until we produced a satisfactorily flat surface. Then we had to figure out how to file the other two sides at the correct angle to complete the triangle.

Once we got the right shape, we tried it in the hole in the head of a screw. Too big! More filing to get the right size. After several steps in this file-and-try process, we finally reached what we were after.

I showed Kyser how to make a wooden handle out of a piece of dowel and how to insert the triangular screwdriver shaft into the handle so it was secure. Then he eagerly began taking the toy apart.

I was fascinated and enjoyed watching this young lad who not only had made his own specialized screwdriver but also had come into intimate contact with the concept of toolmaking. After he had worked for a while, he turned to me and announced that he was going to make several other copies of the screwdriver for his friends, and immediately set about the task with enthusiasm and determination.

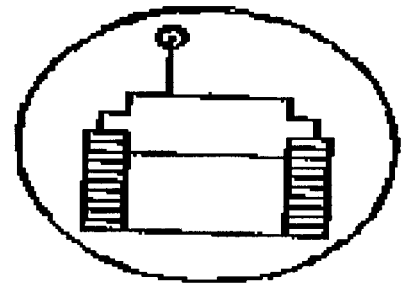
Over the years, several of the *LX* regulars, including Glen Bull and Judi Harris, have emphasized the importance of Logo procedures as tools. Their lessons came rushing back to me as I watched a young boy filing a piece of scrap copper rod, making a tool of his own for a very specialized purpose.

I have seen students create Logo tool procedures of their own to fill special needs or to carry out particular tasks. I have seen students share copies of these tool procedures with others as well, just as Kyser shared copies of his triangular screwdriver.

In both the concrete and in the more abstract manifestations of toolmaking, there is a certain nobleness and a magnificent sense of purpose. I encourage each of you to invite your students to explore this concept.

FD 100!

Tom Lough
Founding Editor
PO Box 394
Simsbury, CT 06070



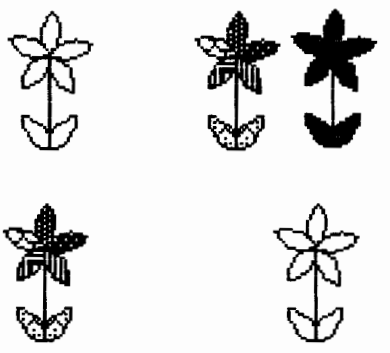
by Geoffrey Yamane

Logo Garden Tools

by Dorothy Fitch

Happy spring! Time to get outside and into the garden. What grows in your garden? Herbs and vegetables? Perennials? Wildflowers? In this issue's editorial I created an extended metaphor about Logo and gardens. For those who wish to create a Logo garden, here are some ideas.

You can use a simple set of arc procedures to make petals (see the Fall, 1993, issue of *LX* for more about arcs.) Put some petals on a stem and you have a flower! Draw lots of them in different colors or patterns and you have a garden. Challenge students who know how to use variables to rewrite these procedures to take an input.



```
TO RARC
REPEAT 2 [R RIGHT 90]
END

TO R
REPEAT 9 [FORWARD 2 RIGHT 10]
END

TO LARC
REPEAT 2 [L LEFT 90]
END

TO L
REPEAT 9 [FORWARD 2 LEFT 10]
END

TO FLOWER
FORWARD 35
RIGHT 180
REPEAT 5 [RARC RIGHT 72]
FORWARD 35
RIGHT 180
RARC
LARC
END
```

Or your turtle can become a flower. Use the shape editor to design a flower head. Draw a stem with the turtle and stamp the flower shape on the screen in rows and columns. Perhaps you want a more casual arrangement. Make different types of flowers. (This may take some research in gardening or wildflower books.)

Terrapin's *Logo PLUS for the Macintosh* comes with several handy multiple turtle tools. (Henri Picciotto, who teaches in San Francisco, provided Terrapin with these clever tools.) LTURTLES sets up turtles in a line.

Here is the result of the instruction:

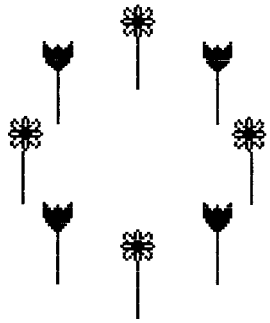
```
LTURTLES 16
```

Now, with just a few instructions, you can turn a row of turtles into a pretty garden.

```
TELL [0 2 4 6 8 10 12 14]
SETSHAPE :TULIP
TELL [1 3 5 7 9 11 13 15]
SETSHAPE :DAISY
PENUP
FORWARD 30
TELL TURTLES
PENDOWN
FORWARD 15
```

CTURTLES points turtles facing out in a circle formation. Typing CTURTLES 8 arranges the turtles like this:

Now it is easy to create a round garden.



```
TELL [0 2 4 6]
SETSHAPE :DAISY
TELL [1 3 5 7]
SETSHAPE :TULIP
TELL TURTLES
PENUP
FORWARD 30
SETHEADING 0
PENDOWN
FORWARD 20
```

How about an animated garden? Use multiple turtles to make your flowers appear to grow. A random instruction ensures that the flowers grow at different rates.

```
LTURTLES 8
SETSHAPE :TULIP
REPEAT 15 [EACH [FORWARD RANDOM 4]]
```

What if the last instruction is

```
REPEAT 15 [FORWARD RANDOM 4]
```

instead? Does it make a difference? Try it and see!

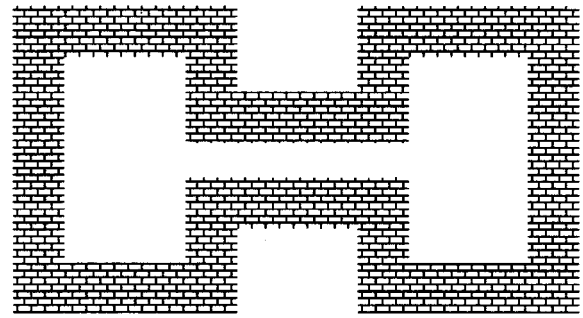
How else could you animate your scene?

- add changing weather:
  clouds drift by
  a rain storm provides water
- show the passing of time:
  the sun travels across the sky
  the background color of the screen changes as the sky gets dark
  stars come out
- have other creatures visit:
  ants crawl along the ground
  a bee buzzes by
  a robin lands on a wire or in a tree

a snake slithers past
a bunny nibbles on a carrot
a hummingbird sips nectar from a flower

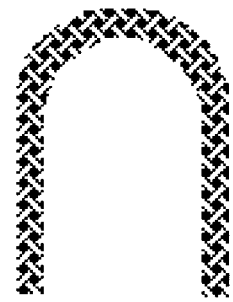
As a bird watcher, I couldn't resist that idea! Switching back and forth between these two shapes makes the hummingbird hover. (In a serendipitous accident, I put the pen's drawing position for the second bird shape in a slightly different place than that of the first. The result is that when I switch between the shapes, the hummer appears to jiggle a little as it hovers—quite a realistic effect!)

- decorate your garden area with a brick walk: Change the pen size and move the turtle for a quick and easy brick walk!



```
SETPENSIZE 30 30
SETPPATTERN 12
FORWARD 50
RIGHT 90
FORWARD 100
RIGHT 90
and so on...
```

or a lattice archway:



```
SETPPATTERN 16
SETPENSIZE 10 10
FORWARD 60
REPEAT 18 [FORWARD 6 RIGHT 10]
FORWARD 67
```

The second side is longer than the first because of the enlarged pen—the pattern extends beyond the turtle's normal drawing position when it moves. To compensate, we can move the turtle a bit farther on the second side.

It's up to you to combine these ideas (and more of your own) into a garden of your design. With shapes, multiple turtles, colors, and patterns, your garden can come alive. Save your program so that you can run it in the winter when you need a breath of fresh air!

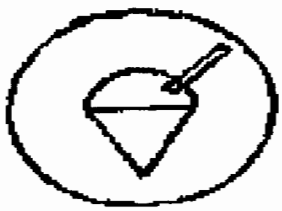
If you would like "fat-bit" diagrams for the shapes used in this column, please send a self-addressed, stamped envelope to LX Shapes, Terrapin Software, Inc., 400 Riverside Street, Portland, ME 04103.

Happy Logo adventures, and gardening!

Dorothy Fitch has been director of product development at Terrapin since 1987. A former music educator, she has also directed a computer education classroom for teachers and students and provided inservice training and curriculum development for schools. She is the author of *Logo Data Toolkit* and coauthor of *Kinderlogo*, a single-keystroke Logo curriculum for young learners. At Terrapin, she coordinates software development, edits curriculum materials, writes documentation, and presents sessions at regional and national conferences.

Dorothy Fitch
Terrapin Software, Inc.
400 Riverside Street
Portland, ME 04103-1068

CompuServe: 71760,366
Internet: 71760.366@compuserve.com
800/972-8200



Crystal Rain Forest—A Learning Adventure

The planet Oglo is in big trouble. Its last remaining rain forest is rapidly disappearing as the greedy Cut and Run Sawmill Gang cut their way through it for profit. The King of Oglo, dismayed by the destruction of his forests, has banned all tree cutting, but the Cut and Run Gang has poisoned him and now he lies dangerously ill in the hospital. Only the children (with a bit of help here and there) can save him, the forest, and, subsequently, the planet from extinction. All they have to do is locate the magic crystals that can be found somewhere deep in the heart of the rain forest.

However, only the elusive Professor Roberts, who lives in Bridgetown, knows the whereabouts of the crystals, and the children are faced with a variety of ingenious puzzles and problems as they try to track him down. When they eventually find their way into the rain forest, the children meet a host of its colorful inhabitants before facing their final challenge in the quest to save the King.

A Mathematical Adventure Into Logo

All the puzzles and problems are related to Logo and are encountered in a carefully structured manner. By the time the children have completed the Crystal Rain Forest adventure, they will have learned how to use Logo without even realizing it. The package even includes a copy of Crystal Logo, an easy-to-use version of Logo that can be used completely separately from the adventure and can be tailored to suit your own needs in the classroom.

A Starting Point for Conservation and Nature Work

The adventure contains numerous starting points for all sorts of work on conservation and nature and is ideal for use as the centerpiece for exploring rain forests.

Great Fun

With its marvelous graphics and humor, the adventure appeals to children and adults alike.

For ages 8 and up. Available in a single version, school version, or with site license. Requires PC-compatible computers, 286 and above, with VGA and a mouse.

For more information, contact: Terrapin Software, Inc., 400 Riverside St., Portland, ME 04103-1068; 1-800-972-8200

Logo Procedures for Paralyzed Children

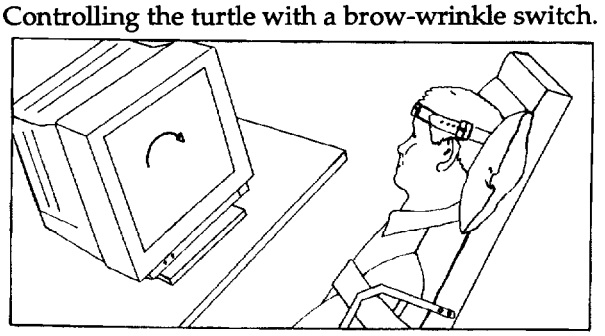
by James Blodgett

Some children suffer such severe handicaps that they cannot draw, point, or manipulate objects. This is especially troublesome in a child because it eliminates many opportunities for learning and relating to the world. Fortunately, a computer can create a world within which all of these things—pointing, drawing, manipulating, communicating—can be achieved by children with severe motor and communication handicaps.

How can a paralyzed child control a world within the computer if he or she cannot use the keyboard? This is done with switches. Few children are so paralyzed that they cannot blink an eye, wrinkle a brow, or make some small movement. Commercial switches are available that can sense these movements. These switches can be connected to the computer so that they substitute for the paddle button, joystick, or mouse button.

How can a paralyzed child control a computer by pressing one or two switches? Imagine a Logo turtle that is rotating clockwise. When the switch is pressed, the turtle stops rotating and begins moving in the direction it is pointing. A child can easily move the turtle to any position on the screen by waiting until it is pointing in the right direction and then pressing the switch to make it move forward. This switch-controlled turtle can be used to draw pictures, a stimulating activity for paralyzed children who would not otherwise be able to draw. The switch-controlled turtle also can control the computer by selecting icons like a mouse pointer. Other protocols allow selection of letters, words, and commands.

Controlling the turtle with a brow-wrinkle switch.



Switches for paralyzed children are called *adaptive switches*. The switch-controlled rotating pointer is called an *arrow cursor*. An arrow cursor is easy to implement in Logo. A short Logo implementation is listed below.

Some children who are paralyzed have multiple disabilities. For example, they may be mentally handicapped, or they may be unable to talk. In these cases, alternative communication is a fundamental need. Mentally handicapped children who have trouble with language may be able to communicate by pointing at pictures on a *communication board*. This is not possible for paralyzed children who are unable to use their hands to point. But Logo can help. It is easy to make a switch-controlled communication board with Logo. The turtle is set to rotate like a clock hand in the center of the screen and is surrounded by pictures or symbols to be chosen. The turtle is stopped by releasing the switch when it points at the desired picture.

These Logo procedures can be dramatically liberating for handicapped children. The procedures give them control on the computer screen that they do not have in the physical world. One handicapped user of a similar program wore out several switches in a short period of time! The Logo procedures also can help a handicapped child demonstrate that he or she can accomplish interesting and useful projects with a computer. Such demonstrations may promote the purchase of commercially available adaptive software and hardware. Use of such software and equipment could prove crucial to the development of a handicapped child. Some handicapped single-switch users have accomplished outstanding achievements. For example, a recent best-selling book was written by a handicapped physicist, Stephen Hawking, using a single-switch word processor.

Switch-Controlled Drawing Procedures

```
TO START
MAKE "AMOUNT 3
CLEARSCREEN
SETHEADING 0
PENUP SETXY -140 120
PENDOWN
REPEAT 2 [RIGHT 90 FORWARD 40 RIGHT
   90 FORWARD 20]
PENUP SETXY 0 0 PENDOWN
ARROW
END
```

```
TO ARROW
IF PADDLEBUTTON 1 THEN FORWARD
   :AMOUNT ELSE TURN
ARROW
END

TO TURN
IF HEADING = 0 THEN OFF
IF PADDLEBUTTON 0 THEN RIGHT :AMOUNT
   ELSE LEFT :AMOUNT
END

TO OFF
HIDETURTLE
REPEAT 360 / :AMOUNT [IF
   PADDLEBUTTON 1 THEN
   SWITCHPENSTATE]
SHOWTURTLE
END

TO SWITCHPENSTATE
IF ITEM 1 TURTLESTATE THEN PENUP
   ELSE PENDOWN
IF ALLOF XCOR <-100 YCOR> 100 THEN
   START
END
```

There procedures are written in Terrapin Logo. For Commodore Logo, replace PADDLEBUTTON with JOYBUTTON and replace TURTLESTATE with DRAWSTATE.

How to Use the Switch-Controlled Drawing Procedures

Begin by typing START. The turtle will rotate to the left. When you press paddlebutton 1, the turtle stops rotating and moves in the direction it is pointing. When the turtle is pointing up, it disappears briefly. Pressing paddlebutton 1 while the turtle is not visible toggles between PENDOWN and PENUP. When the pen is down, the turtle can be used to draw lines.

These procedures make drawing easy. Using them is also fun for children who are not paralyzed. Graceful curves and spirals can be drawn by toggling rapidly between the movement and the turning phase. The procedures can be used with only one switch, but a second switch, paddlebutton 0, if the user can handle it, adds more control. The second switch changes the direction the arrow is turning. This allows lines to curve in more than one direction. With careful control, intricate pictures and even signatures can be executed. The following picture was drawn using these procedures and traced from the screen:

A switch-controlled turtle drawing.

You can test these procedures without an adaptive switch. On the Apple, press Closed-Apple (or Option) to move the turtle forward and Open-Apple to change the direction in which the turtle turns. In Commodore Logo, the joystick can be replaced by the space bar as one button, and the combination of the RUN/STOP key and the N key can be used as the second button.

An arrow cursor can control the computer by selecting icons, like a mouse. This implementation has a single icon, an eraser, the rectangle in the upper left of the screen. The user can erase the screen by moving the turtle into the eraser area and then "clicking" on it by pressing the switch when the turtle disappears.

You may want to change the speed of the turtle movement. Some computers run so quickly that the procedure as listed will be much too fast. Also, some handicapped people have a slow response time. They can control the turtle only if it moves slowly. Finally, there are artistic reasons for different turtle speeds. A slow turtle can execute careful detail; a fast turtle draws with swift broad strokes.

The speed of turtle rotation and movement can be slowed by making the value of the variable AMOUNT equal to a lower number, either 1 or 2, in the START procedure. The turtle can be made to go faster by making AMOUNT a higher number as long as it is an integer that divides 360 evenly. The speed of the two aspects of turtle movement, moving forward and turning, can be adjusted individually by modifying the program appropriately at each place that AMOUNT appears. If you need to make a large adjustment in the speed of turtle movement, you may need to add a WAIT command (such as WAIT 3) just before the ARROW statement at the end of the ARROW procedure. If your version of Logo does not have a WAIT command, use this procedure.

```
TO WAIT :NUMBER
IF :NUMBER = 0 THEN STOP
WAIT :NUMBER - 1
END
```

Adjust the value passed to NUMBER and keep testing the procedure until it slows the computer the desired amount.

Note that the program can be stopped by pressing Control-G.

Controlling the turtle seems simple, but paralyzed children may have trouble with it. They are not used to manipulating objects. Several sessions of practice in controlling the turtle may be necessary before the children can begin drawing what they want in Logo.

A Very Short Drawing Procedure

```
TO ARROW
IF PADDLEBUTTON 1 THEN FORWARD 3
   ELSE RIGHT 3
ARROW
END
```

This procedure implements a basic single-switch arrow cursor with a minimum amount of program entry. It is useful to demonstrate the basic arrow. It can be used to teach a handicapped child who has not yet learned other functions. It is also a good starting point for persons writing their own arrow cursor procedures, since it shows how easy it is to implement a basic arrow cursor in Logo.

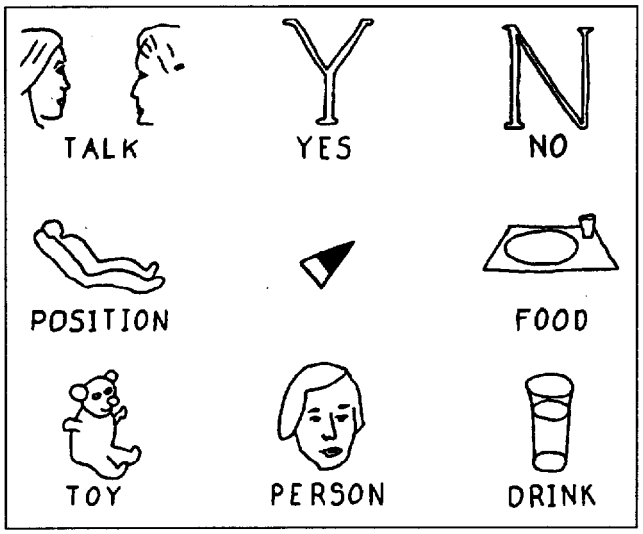
Note that this can be started with PENUP or PENDOWN. The speed can be changed by changing the two 3s. Also, see the comments for converting to the Commodore, stopping the program, and adding a WAIT procedure in the instructions for the first set of procedures.

A Communication Board

```
TO POINT
IF PADDLEBUTTON 1 THEN RIGHT 3
POINT
END
```

This procedure makes the turtle turn when the switch is pressed and stop when the switch is released. This is useful in a communication board in which the turtle is used to select pictures and symbols, as in the accompanying figure. Symbols should be individualized for the user. The symbols can be drawn by the turtle, or they can be drawn by hand on a sheet of transparent plastic and taped over the screen.

Refer to the comments on changing speed, Commodore conversion, paddlebutton equivalents, and stopping the program in the preceding instructions.



A communication board.

A Larger Communication Board Arrow

```
TO START
MAKE "AMOUNT 3
PENUP HIDETURTLE
SETXY 0 0
SETHEADING 0
TURN
END

TO TURN
PENCOLOR 1
ARROW
REPEAT 2000 / :AMOUNT [ ]
IF NOT PADDLEBUTTON 1 THEN PENCOLOR
   0 ARROW RIGHT 45
TURN
END

TO ARROW
LEFT 90
FORWARD 3
PENDOWN
FORWARD 9
RIGHT 135
FORWARD 17
RIGHT 90
FORWARD 17
RIGHT 135
FORWARD 9
LEFT 90
FORWARD 7
RIGHT 90
FORWARD 6
RIGHT 90
FORWARD 7
PENUP
SETXY 0 0
END
```

The Logo turtle may be difficult to see for some users. This procedure draws a larger arrow. (In some versions of Logo you can also create an arrow in the shape editor.)

Begin by typing START. The arrow turns while the switch is pressed. Note that if you start from a text screen, you will have to press the switch to see the arrow.

This arrow is set to point at eight symbols located at the points of a compass, that is, north (up), northeast, east, and so forth. The number of symbols can be changed by changing the angle of rotation in the TURN subprocedure from 45 to the value of 360 divided by the number of symbols.

The speed that the arrow turns can be changed by changing the value of the variable :AMOUNT in the start procedure. In this case, AMOUNT can be greater or less than one.

There are two changes needed in this set of procedures to convert it for Commodore Logo. One is to change PADDLEBUTTON to JOYBUTTON. The other is to change PENCOLOR 0 to PENCOLOR -1 in the TURN subprocedure. Note the comments on the paddlebutton keyboard equivalents and stopping the program mentioned previously.

If you want to run a procedure when the arrow is pointing toward a particular symbol, you can change the procedure TURN as follows, and then add a procedure CHOOSE, where subprocedures are run based on the turtle's heading.

```
TO TURN
PENCOLOR 1 ARROW
REPEAT 2000 / :AMOUNT [
IF NOT PADDLEBUTTON 1 THEN PENCOLOR
   0 ARROW RIGHT 45 ELSE CHOOSE
TURN
END

TO CHOOSE
IF HEADING = 45 THEN ACTION1
IF HEADING = 90 THEN ACTION2
 and so on...

END
```

(Replace ACTION1 and ACTION2 with the procedure names that should run if these icons are selected. The heading numbers you use may be different.)

It is exciting to realize that with the addition of adaptive switches and the few Logo procedures described here, children and adults who are paralyzed can be empowered to communicate and create. These and other adaptive technologies offer a new, accessible world for communication among individuals with handicapping conditions.

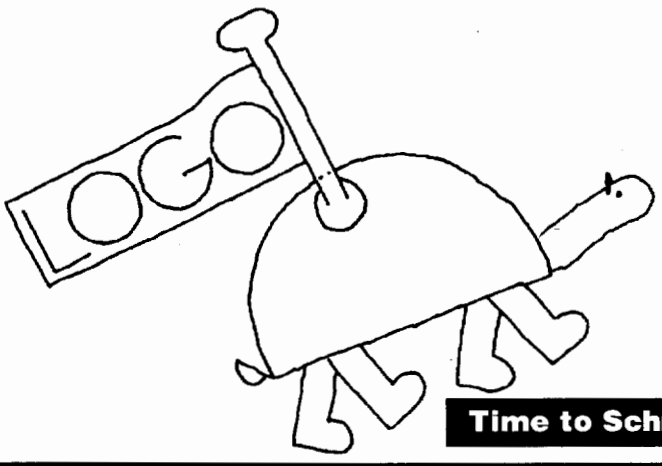
The following publications are sources of additional information about adaptive switch products:

Closing the Gap
PO Box 68
Henderson, MN 56044
612/248-3294

Communication Outlook
Artificial Language Laboratory
Michigan State University
405 Computer Center
East Lansing, MI 48824-1042
517/353-0870

James Blodgett, president of Adaptive Computers, develops and markets adaptive switch software. He designed these Logo procedures to help handicapped children when funds are not available to purchase commercial software. An earlier version of this article was published by Terrapin Software, Inc. Dorothy Fitch of Terrapin contributed several improvements to the procedures.

SAVE SUNDAY, JUNE 12, 1994
FOR THE LOGOSIUM (LOGO SYMPOSIUM)



Mini How To Do It Sessions

Jitterbug Logo Style—It's Procedural!

Students, Student Projects, and Videotapes

Time to Schmooz With Logo Users From Around the World

Interactive, Informal Group Discussions on Logo Topics—Assessment, NCTM Standards, Action Research, New Environments, Beginning and Advanced Topics...

Sponsored by ISTE's SIGLogo and the Logo Foundation ■ Hosted by The Media Lab, M.I.T.
A one-day pre-conference at NECC '94, Boston, Massachusetts
For NECC '94 Advance Program, contact NECC '94, 1787 Agate Street, Eugene, OR 97403-1923

A First Course in Programming
in Terrapin Logo, LogoWriter, and PC Logo

This is a complete curriculum for a semester course in programming. It includes student activity sheets, teacher lesson preparation sheets, tests, quizzes, assignments, and sample solutions for all student assignments (hard and softcopy!)

A First Course in Programming is a directed learning environment in structured programming. Its 450 pages emphasize proble solving stategies, critical thinking skills, and solid principles of computer science.

Only $150 for a building site license. Call us for further information.

Curriculum written BY teachers FOR teachres!

Logo Curriculum Publishers
4122 Edwinstowe Avenue
Colorado Springs, CO 80907
1-800-348-5646 (FIT LOGO)

Junk Mail—It's All Around Us

by Robert Macdonald

Musings

In late September several years ago, a young girl who was newly enrolled in the fourth grade rushed into my classroom before school began, inordinately excited over a book she had cradled in her arms. The book was a personalized volume that her younger brother had received for his birthday. It was similar to books that a publisher had offered to our students during the annual November book fair the previous year.

I was charmed by her enthusiasm. A few of her peers couldn't quite share her enthusiasm. After all, one remarked, "You get stuff like that through the mail all the time."

As class began we discussed personalized books. I asked, "How do you think it was done?" From the experiences of previous years, some students remembered that they had filled out a sheet that had requested their name, address, age, birthday, pet, the names of brothers and sisters, and so forth. Obviously, one student observed, someone used this material to create the personalized book.

Several weeks before, I had introduced my students to an interactive program in *LogoWriter* in which the computer had asked them for information and then went out of its way to bring to the students' attention the little-known foibles of their teacher. Using this as a point of departure, I was able to pin-point some of the ways in which a book manufacturer could do much the same thing to capture the interest of a reader. After all, we all like to see our name in print. More than that, in the work world we all enjoy those moments of light relief from regular chores.

An Interactive Program

The following is an interactive program simulating a brief conversation with a computer. The program is written in *LogoWriter*. It is similar to material I like to provide students on an individual scrapbook disk early in the year. After calling their attention to certain features, some of which I will remark on below, they are free to change the content in any way that appeals to them. Patterning new programs after pre-existent programs is an excellent way for students to gain mastery over a somewhat complex topic. The command for running the program is CONVERSATION.

```
to conversation
introduction
chit.chat
talk.back
agree.disagree
finish.up
end

to introduction
clearpage
spaces 2
print [Well, HELLO THERE! I am Mr.
   Computer.  Who are you?  Please
   type in your name.]
spaces 1
make "name first readlist
spaces 2
(print [I'm glad to meet you,] word
   :name ".)
spaces 2
(print (Would you mind patting the
   top of my head,] word :name ", [I
   feel a little strange up there.])
spaces 2
wait 100
end

to clearpage
if not front? [flip]
rg
ht
ct
cc
end

to spaces :number
repeat :number [print []]
end
```

```
to chit.chat
clearpage
spaces 2
print [Please write something for me
   to say.]
spaces 1
make "writing readlist
spaces 2
(print [You just forced me to say]
   :writing)
wait 100
end

to talk.back
clearpage
spaces 2
print [Please type something else
   for me to say.]
spaces 1
make "reply readlist
spaces 2 (print [But I hate to say]
   :reply)
spaces 1
(print [Gosh! You really are a
   nerd,] word :name ".)
wait 90
end

to agree.disagree
clearpage
spaces 1
print [Tell me something that you
   like.]
spaces 2
make "like readlist
spaces 2
(print [I like] :like [too!])
spaces 2
wait 25
print [Tell me something that you
   hate.]
spaces 1
make "hate readlist
spaces 2
(print [I hate] :hate [even more
   than you do.])
wait 90
end

to finish.up
clearpage
print [I must be on my way. I have a
   funny feeling that Mr. Macdonald
   knows that we are fooling
   around.]
spaces 1
wait 60
(print [Come back again some day,]
   word :name ".)
wait 60
spaces 1
print [Why are you still here?
   You'll have the old boy on the
   warpath.]
wait 70
spaces 1
clearpage
repeat 3 [spaces 10 print [GOOD-BYE!
   ! ! ! ! ! !] wait 30]
clearpage
end
```

Examining the Program

Before discussing the above program with students, some prior knowledge of graphics, superprocedures, main procedures, variables, and top-down design might be necessary (see Yoder, 1990). Or you can discuss only the things that you feel the students might have an interest in at that moment. They will absorb what they need as they are ready for it.

A few remarks about the print statements are essential. In *LogoWriter,* the use of parentheses will be mandatory if you are inserting a number of separated bracketed materials. Consequently, note:

```
(print [stuff] [more stuff] [still more
   stuff])
```

If you do not use the primitive **word** as a form of "glue," you will have unpleasant spaces between a word and the following punctuation. Delete it in one of the print statements and you will quickly discover what I mean. For example, in the Command Center write

```
make "name "Veronica
```

and then press Return. Then type

```
(print [I'm glad to meet you,] :name ".)
```

and press Return.

On the page you see:

```
I'm glad to meet you, Veronica .
```

The period appears after a space. If this bothers you (as it does me), you'll want that space to disappear. You'll need to glue the period to Veronica. Hence, in the Command Center, type

```
(print [I'm glad to meet you,] word
   :name ".)
```

and press Return. The need for this "pasting" is a disconcerting little problem that is related to the way the Logo language places spaces between elements in a **print** statement. Using **word** is one way to fix the problem.

In the preceding program, I used the primitive **make** to assign a variable name, such as name, writing, and **reply**, and to give it a value. This is particularly helpful if you wish to use an assigned value several times in the same program. I could have used the primitive **name**, whose function is identical to **make**; however, the inputs are provided in reverse order. I thought that was just a bit awkward for my purposes. Thus, I will assign variables in a different fashion in the following **junkmail** program.

Preparing a Junk Mail Microworld

With the previous experience of my **conversation** program in mind, it didn't take very long for the class discussion to be expanded to junk mail. *Webster's Dictionary* (1984) defines "junk mail" as "third class mail (as advertising circulars) that is often addressed to 'occupant' or 'resident'." The term "junk mail" was entered in Webster's lexicographical list in 1954. We all know how highly personalized it has become since.

Junk mail promoters always have some type of form letter in which are inserted all of those personal touches that help warm us up to the coming pitch. The best way of realizing this is to have a class collect as much junk mail as possible and examine it. It doesn't take very long for an enterprising class to flood a classroom with the material.

To provide a program for a class to experiment with junk mail, you'll have to arrive at a topic acceptable to a majority of the class or be prepared to do a lot of program editing. As I recall pets, candy, magazines, and breakfast cereals headed the list. Breakfast cereals won out. Then we had to agree on what variables we would input. We settled on our own name, our address, a favorite friend, an imaginary cereal, and a tempting flavor. Then the teacher went to work.

The Junkmail Program

This program was also written for *LogoWriter*. Some interesting aspects of the program are discussed following the program. The command for running the program is **junkmail**.

```
to junkmail
clearpage
message select.name select.address
   select.friend select.cereal
   select.flavor
end

to clearpage
if not front? [flip]
rg
ct
ht
cc
end

to select.name
clearpage
print [Please type in your first and
   last name.]
output readlist
end

to select.address
clearpage
print [Please type in your street
   address. For example: 725 Main
   Street.]
output readlist
end

to select.friend
clearpage
print [Please type in the name of
   your best friend.]
output first readlist
end

to select.cereal
clearpage
print [Please type in the trade name
   of an imaginary cereal.  Make it
   a single word.]
output first readlist
end

to select.flavor
clearpage
print [Please type in your favorite
   flavor.  Also make it one word.
   For example: vanilla, nutty,
   strawberry, avocado.]
output first readlist
end

to message :name :address :friend
   :cereal :flavor
clearpage
print []
print :name
print :address
print [Grosse Ile, Michigan 48138]
print []
print []
(print "Dear word first :name ",)
print []
```

```
paragraph1 :friend :cereal
paragraph2 :friend :cereal
paragraph3 :flavor
paragraph4 :cereal
print []
print [Sincerely Yours,]
print []
print []
(print [The World of] :cereal)
publish
end

to paragraph1 :friend :cereal
tab (print [Congratulations !!! You
   and your friend,] word :friend ",
   [have been selected as possible
   winners of a year's supply of]
   word :cereal ".)
end

to paragraph2 :friend :cereal
tab (print [All that you and]
   :friend [need to do is write in
   25 words or less what you like
   best about] word :cereal ".)
end

to paragraph3 :flavor
tab (print [Your choice of] :flavor
   [as a flavor surely demonstrates
   your good taste and may well
   indicate a winner.])
end

to paragraph4 :cereal
tab (print [Please inform your
   parents to prepare a large space
   for your new supply of] word
   :cereal ".)
end

to publish
type (if you wish to print this
   message, please touch the CONTROL
   KEY and P.]
when "p [sspace printtext]
end
```

By following the prompts given by the above program, you might get a letter such as this:

```
Bobby Miller
121 South Nottawa Street
Grosse Ile, Michigan 48138

Dear Bobby,

Congratulations!!! You and your
friend, Jim, have been selected as
possible winners of a year's supply
of NuttyOats.
All that you and Jim need to do is
write in 25 words or less what you
like best about NuttyOats.
Your choice of nutty as a flavor
surely demonstrates your good taste
and may well indicate a winner.
Please inform your parents to
prepare a large space for your new
supply of NuttyOats.

Sincerely Yours,

The World of NuttyOats
```

Examining the Junkmail Program

I decided to handle my variables in junkmail in a markedly different fashion. In order to provide variables for **name, address, friend, cereal,** and **flavor** in the procedure **message**, I wrote other procedures to receive values from the computer user and then output those values. Hence, the procedures **select.name, select.address, select.friend, select.cereal,** and **select.flavor** output values decided upon by the user of the program. Thus these procedures act as reporters of information.

It might prove beneficial for the reader to trace the movement of this information through the program. From the "select" procedures, we move on to the procedure **message,** which on its title line has five inputs. Within the **message** procedure we find that four **paragraph** procedures are called, each of which passes information.

In conclusion, you might notice the application of a **when** command in the procedure **publish**. This provides an easy way to introduce programmable keys used in conjunction with the Control key—a fun way to get a printed copy of the letter.

The program is open to all sorts of modifications, either by a teacher or by students. The possibilities appear endless. Isn't it a joy to acknowledge that even junk mail has its positive side?

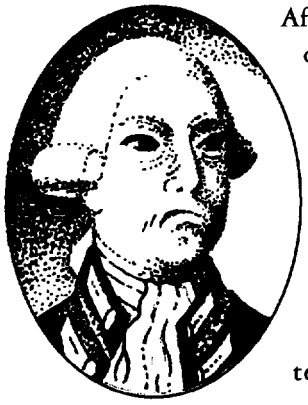
References

Yoder, Sharon. (1990). *Introduction to programming in Logo using LogoWriter* (rev. ed.). Eugene, OR: International Society for Technology in Education.

Webster's Ninth New Collegiate Dictionary. (1984). Springfield, MA: Merriam-Webster.

Robert Macdonald
Hawthorne Meadows
10225 Nancy's Blvd., Unit 63
Grosse Ile, MI 48138

KING GEORGE WOULD HAVE HATED THIS CONFERENCE...



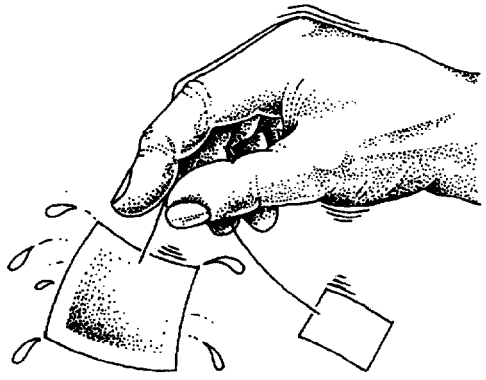
After all, it's going to be everything he was against...new ideas, new thinking, new challenges—*Revolution!*

Taking place in the historically non-traditional city of Boston, Massachusetts, The 1994 National Education Computing Conference will offer possibilities for exploring educational uses of technology that have never been more exciting.

The special focus for '94 is on educators who have been using technology to create *revolutionary* learning environments in their classrooms. We also hope to provide participants with more hands-on opportunities to experience groundbreaking technologies than you can shake a tea bag at!

Session topics include:

- multimedia and distance learning,
- uses of computer-based technologies at all academic levels and in all academic disciplines,
- integration of technology into the curriculum,
- technology and school restructuring,
- teacher education,
- technology and the arts,
- and new and emerging technologies for instruction and information management.



The conference also boasts one of the largest and most progressive trade shows of its kind. Nearly 500 exhibitors will be on hand to give participants the chance to explore the latest technologies up close and personal.

With NECC '94, we invite you to share in *Recreating the Revolution*—and to recognize that nothing short of an educational revolution will provide the new basic skills necessary for the Information Age.

Besides...we think Paul would have *loved it!*

For more information on attending workshops, registering, and finding accommodations contact:
Donella Ingham, NECC '94
1787 Agate Street, Eugene, OR 97403
(PH) 503/346-2834
(FX) 503/346-5890
(IN) donella_ingham@ccmail.uoregon.edu

Recreating the Revolution
NECC '94
Boston

John B. Hynes Veterans Memorial Convention Center, Boston, MA
June 13-15, 1994

Hosted by Lesley College and Bolt, Beranek & Newman, Inc.

Graphics in This Issue: LogoPOGS

by Sandra Siu

In Hawaii, students have gone crazy over POGS! As a teacher who has taught Logo to students in grades 4-6 for the last 8 years, I wondered what better way there was to motivate students and integrate computers into the curriculum that to have them make LogoPOGS.

In case you don't know what POGS are, let me explain. POGS are printed milk cap covers and their look-alikes. They are round in shape. They are collected by young and old alike and range in price from 10 cents to 5 dollars for artist-designed POGS.

Children play the milk cap cover game by doing this:

1. First they find a floor or flat surface.
2. Then they stack a few POGS neatly in a pile.
3. Each player then takes turns aiming at the pile with a POG (called a "hitter").

With enough force at the proper angle, the stack is scattered. The player gets to keep any of the POGS that land upside down. The name POG originated from the Passion-Orange-Guava drink put out by Haleakala Dairy, which resurrected the game 20 or so years ago when it gave away caps as a promotional giveaway.

I am currently the enrichment teacher at Kahalu'u Elementary School in Kaneohe, Hawaii. Our school is located in a rural community. The student body consists of multiethnic groups—Hawaiian, part-Hawaiian, Filipino, Oriental, and Caucasian. Academically, our students fall in the average and below-average groups, with a few high-average students. Some own Nintendo sets, but few own a computer.

Because at one time I worked as the computer teacher, sometimes teachers ask me to teach their computer classes for them. This year I worked with a class of 28 sixth-graders, some of whom had experienced Logo before. We started by reviewing the basic commands and discussing shapes, angles, and turns. We used geometric terms such as parallel, perpendicular, and diagonal. We identified right angles, acute angles, and obtuse angles. We practiced drawing pictures by using geometric shapes. Then we practiced making procedures for simple pictures, such as a sailboat on the water. I emphasized looking at the shapes and then making a procedure to make each shape.

At first, there was a wide difference in the problem-solving ability of the students. Some children started with the circle and then tried to make their design one step at a time. Others were able to make simple procedures for each shape but had trouble putting them together. A couple of students made the design first and then added the circle. A few children tried to make the whole picture in one procedure. Some had no trouble at all. To help those having difficulty and also to broaden their understanding, I found that having the whole class work together on one design at first helped them gain a better understanding of procedures and superprocedures and helped them to work more efficiently when they worked independently again.

After about eight lessons of basics, experimenting, and practice, I gave the assignment to design a POG. Students were encouraged to start with simple designs that used definite shapes. The students worked in pairs.

Apple Logo was used with a program called LOGOGRAF Picture from the Apple Logo Toolkit to print the pictures. All students who worked on the project successfully completed one or two POGS. Most

were able to program, debug, and reprogram whenever necessary. Some of my observations were:

1. Students worked well together and tried to help each other.
2. They tried different ways to do things.
3. If they were unsuccessful, they tried again.
4. They showed excitement when something turned out right.
5. Some groups included children who "took charge."
6. Most groups shared the responsibility equally.
7. Even though you give suggestions to students, they don't always take them—they want to find out in their own way.
8. Most students did not ask for help. They tried to solve their problems by themselves.

At the end of the project, almost all of the students reported that they thought that Logo was "fun, educational, and challenging." A few said that they sometimes felt frustrated when their design didn't come out right, but they all said that this made them determined to fix it and solve their problems. They were all pleased with their LogoPOGS, which were displayed on a large POG-shaped poster paper.

From a teacher's point of view, I think Logo was the perfect tool to integrate geometry, problem-solving, programming, and art skills.

Sandra Siu is the enrichment teacher at Kahalu'u Elementary School in Kaneohe, Hawaii. She has previously taught regular classroom grades K-4, computer classes for grades 4-6, and language arts classes for grades K-3. She has taught Logo for more than five years and thinks Logo is one of the best programs for teaching problem-solving to children.

Mrs. Sandra Siu
45-606 Kulukeoe Street
Kaneohe, HI 96744
Internet: SIU@UHCCVX
Phone: 808-247-0262

A Case of Curiosity: Part Two

by Eadie Adamson

> A small but crucial part of discovery of the highest order is to invent and develop effective models or "puzzle forms."... It is only through the exercise of problem solving and the effort of discovery that one learns the working heuristics of discovery; the more one has practice, the more likely one is to generalize what one has learned into a style of problem solving or inquiry that serves for any kind of task encountered—or almost any kind of task.
>
> Jerome Bruner,
> "The Act of Discovery," *On Knowing*

In the previous issue of *Logo Exchange* (Volume 12, Number 2), I discussed a number of tools for exploring, evaluating, or debugging a *MicroWorlds* Logo project. The ideas came up in the process of enlarging the Maze.Maker project that is contained in the Going Further folder in *MicroWorlds Project Builder*. What follows is a discussion of the paths I traced as I made a large-scale Maze.Maker for a visually impaired child to use.

A First Stab at Figuring Out the Maze.Maker

Here are my first notes, which I added to the original Procedures Page in order to print them:

```
Maze.Maker
Individual turtle pieces are programmed:
    pick-and-place "shape-name
Drawer turtle (??) is programmed:
    adjust stamp ht
Drawer turtle is hidden
Turtle shapes have this balloon message:
    Click on me to get a piece like this one.
    Then drag the piece into place and
        click on it again to adjust it.
Drawer turtle has this message:
    My name is: Drawer
    Drag me into place and click on me
        again to adjust my position.
Turtles are frozen in place.
```

How Did I Know?

How did I know about freezing turtles? This is a feature I had used when I worked on setting up the *MicroWorlds Language Art* projects. The command **freeze** can be used to lock turtles, buttons, sliders, and text boxes into place. Then they cannot be moved or changed accidentally by the user of a project. Although it may not be mentioned in any of the projects, the command *is* explained fully in the Vocabulary on the Help menu. **Freeze** takes as input either a single item or a list of turtles, text boxes, buttons, or sliders. Its opposite command is **unfreeze**.

Enlarging the Maze.Maker Project

First, I chose a new project and selected a new size: **giant**. Later I might have to enlarge the project still further (which would mean creating the project again!), depending on how the size worked on my student's large monitor. There is a term that may be used to change the screen size: **newprojectsize** takes two inputs for the horizontal and vertical size. As with the choices from the menu, **newprojectsize** may be invoked only with a fresh project. It may be useful to you if you have odd-sized monitors and want to make a full-screen display. For my student, who has a 21-inch monitor, I found later that I wanted to make the screen even larger. (This meant creating the project once again!) In general use, however, the three sizes from the menu should be sufficient.

Copying the Shapes

I realized that I needed to duplicate the 15 maze shapes in order to produce a similar maze. I had already observed that in the larger project the turtle came up at the default size, 40, while the turtles in the sample were set to size 20. How did I know that? When I was in the sample project, I typed

```
show size
```

in the Command Center. The answer I got was 20. I wanted the turtles to be larger so that they would be easier to see. (A smaller size like 20 on a giant screen might be good for making a more complex maze, however.)

What did I already know about copying shapes? Well, you can click on a shape and press Command-C

to copy (the same effect as pulling down the **Edit** menu and choosing **Copy** from there). I also knew I could paste a shape in the Scrapbook, which made the transferring to the new project a little faster. The shapes were copied and, an added bonus, when they were added to the new project their names came along too! Since the project screen was small and my monitor was large, I could leave the Scrapbook out and move back and forth from the project shapes to the Scrapbook as I copied and pasted the shapes.

A Little Redesigning

I made some decisions about the enlarged project before I continued:

- I didn't want the shapes all in a square, because that would take up too much screen space. I elected instead to put them across the bottom of the page.
- I also decided, for the moment, to leave out the instructions. They could be added, if necessary, on another page.
- Since my student didn't have a color printer (nor did I), I elected to change the shapes to black and white. This simply involved editing each shape and using the paint bucket in the drawing tools to fill areas with black or white.

More Detective Work

After the shapes were made, I had to do some more investigating. The pieces seemed to be turtles, but when I tried to drag them with the mouse, they didn't move. Aha! I knew that turtles could be frozen in place. So there were 15 turtles, each having one of the maze shapes. One at a time I followed this process (notice that I left the first turtle alone here):

- Click on a shape.
- Click on the Turtle Tool.
- Click on the screen once to set a turtle, and click a second time to give it the shape.
- Continue by clicking on the next shape.
- When all 15 turtles were ready, I arranged them across the bottom of the screen.
- Now I had to freeze all 15 turtles. Here's the command I used:

```
freeze [t2 t3 t4 t5 t6 t7 t8 t9 t10 t11
t12 t13 t14 t15 t16]
```

The first turtle would be the maze-making turtle. How did I know that? Well, at first it was a guess, but it turned out to be right.

Before I did any more on the new maze, I went back to the Maze.Maker project, chose the procedures page, selected and copied the two (only two!) procedures that were there. I pasted these onto the Procedures Page on the new, enlarged maze project. Now I had to figure out what they did, and how they did it.

A little knowledge helps. I knew that turtles could be programmed so that a procedure would run when a turtle was clicked on. I looked at the first maze turtle shape's instructions by clicking on the hatcher and then on the turtle:

```
pick-and-place "edge1
```

Well, now I knew a little more. **Edge1** was the name of the shape the turtle was wearing. **Pick-and-place** was one of the two procedures I had copied. **Pick-and-place** used an input, the shape name. (See the procedure at the end of this article.)

I switched back to the enlarged project and programmed the first turtle by doing this:

- Click on the hatcher
- Click on the turtle
- Type the instructions in its box
- Click on OK

I realized that I could also copy most of the instruction—all but the shape name—and then just paste that part in the instruction box of the next turtle. I went back to the first turtle and copied the part I wanted. Then I programmed the rest of the turtles, sometimes stopping to be sure I had the right shape name.

A Little More Polishing

There was one other task I left for last: I clicked on Help (you can use the pull-down menu or just press Command-H). Each turtle had a balloon with a message about what to do. The maze turtles all had the same message:

- Click on me to get a piece like this one.
- Then, drag the piece into place and click on it again to adjust it.

I copied and pasted the messages into balloons on my new project.

The other turtle was hidden until it took on a shape. I clicked on a shape and then turned on Help to see what the turtle said:

- My name is: Drawer
- Drag me into place and click on me again to adjust my position.

Give the Turtle a Name!

Now I had one other slight complication: there was a mysterious turtle named **"drawer"** in the project. How did the first turtle get to be named **"drawer"** instead of simply **t1**? It's an easy answer, but not necessarily obvious. When you click on a turtle, a dialog box comes up. *MicroWorlds* assumes you want to program the turtle, so the instruction line is highlighted. However,

you can also click on the box that identifies the turtle, select the text, and rename the turtle. Thus, all your turtles can have names instead of numbers (just think of the possibilities here!).

I named the drawing turtle **"drawer"** just to keep things the same as in the original project. Then I copied the message from the original Project and pasted it into my new project.

When I tried out the Maze.Maker, I found that the turtle shapes didn't jump forward far enough. I looked at the **pick-and-place** procedure and saw why: the turtle size on the small project was 20; the new size for turtles was 40. **Pick-and-place** has the drawer turtle jump forward 22 steps. I just needed to change the forward command from 22 to 42! For the moment, I left the **adjust** procedure alone. That turned out to be a good decision because it caused no problems.

Ready to Print?

Now it looked as if the maze project was coming together. I was ready to try printing but I didn't want to see the shapes at the bottom of the page when I printed out a maze. The maze-part turtles are frozen, which prevents them from being moved, but they can still respond to certain commands—**ht** is one of them. I wrote a **printit** procedure to hide the shapes, print the page, and then let the shapes show again:

```
to printit
ask [t2 t3 t4 t5 t6 t7 t8 t9 t10 t11
   t12 t13 t14 t15 t16] [ht]
printpage
ask [t2 t3 t4 t5 t6 t7 t8 t9 t10 t11
   t12 t13 t14 t15 t16] [st]
end
```

Then I decided to move the shapes a little closer together so I could fit a button for printing on the bottom also. I had to **unfreeze** the list of turtles. I copied the list from the procedure I'd just written. In the Command Center I typed **unfreeze**, left a space, and then pasted the list of turtles and pressed return. After I moved the turtles into new places, I went back to the Command Center, deleted the letters **"un"** from the **unfreeze** command, and froze the turtles into their new shapes.

A Few Additions

This project was coming along nicely! What else could I add? I decided to make a NewMaze button because there was room for more buttons on the right side of the screen. NewMaze needed to remove the graphics. **Clean**, which clears only the graphics and leaves turtles alone, was the best command to use. The button could have said **"clean"** but I decided that a user should know that they wanted a new maze before clicking that button. A NewMaze procedure whose command was simply **"clean"** was the thing to use here.

Figure 1 shows what the part of the finished project looked like, with all the shapes arranged in a line across the bottom of the page, buttons added to the right. The ? button just goes to an Instructions page.

New Puzzles for New Learning

Thinking about programming is quite different in this environment. Writing procedures is now not the only way to make interesting things happen. Turtles and buttons can have processes defined without writing procedures for them. Procedures may be written when the instruction is long or complex or needs to use a variable (as in the **pick-and-place** procedure in the Maze.Maker). The procedures will not necessarily reveal everything about a project, although they may yield some clues. In fact, some projects that appear quite complicated may need no procedures at all: not only turtles but also colors can be programmed. Even buttons can be just simple Logo instructions. The Help balloons can be set up to give instructions or clues for the user.

I saw this project as a fascinating puzzle. Armed with a little knowledge about the things to explore, I was able to figure out how the Maze.Maker worked. I made sense of it in figuring it out *for myself*. I knew what tools to use, of course, but I had to "reinvent" the project in order to reproduce it.

Teasing out the processes involved in such a project is a challenging pursuit. The exercise seems to me to be an interesting way to teach kids, who have a natural curiosity anyway, about how to use *MicroWorlds* Logo. Because not all is revealed through procedures, it may take some time and ingenuity to tease out what's making things happen. Presenting a *MicroWorlds* project as a puzzle to figure out how it works is a wonderful challenge. The task of figuring out how a project sample works, re-creating it, and then adding one's own touches extends the learning environment still further. As James Clayson said in another context in his *Visual Modeling with Logo*, "It's the tinkering that counts" (p. 16).

Figure 1.

My explorations were fun and interesting for me, but they set me thinking again about learning. Too often in school we learn just by being told how to do things. We follow instructions, sometimes adding twists of our own. Occasionally we're asked to try out new things to notice how they work. Rarely is an example given as a puzzle to solve. Only infrequently are kids challenged to be explorers. *MicroWorlds Project Builder* gives us some ready-made puzzles for kids (and teachers) to explore. Students need to know how to use some exploring tools, such as those listed above. Then they can try to solve the puzzle of "how it works" for themselves. What an interesting way to learn!

If you use the samples for exploring, I'd love to know how students responded. If you or your students develop any new projects based on these ideas, I'd love to hear about them. Please send samples!

Below are the procedures I used in my expanded Maze.Maker. Only the **adjust** and **pick-and-place** procedures are used in the sample project.

```
to adjust
setx 20 * int round xcor / 20
sety 20 * int round ycor / 20
end
```

The forward instruction here is for a turtle whose size is 40:

```
to pick-and-place :shape
let [here pos]
ask "drawer [ht setpos :here forward
   42 setsh :shape st]
end

to printit
ask [t2 t3 t4 t5 t6 t7 t8 t9 t10 t11
   t12 t13 t14 t15 t16] [ht]
printpage
ask [t2 t3 t4 t5 t6 t7 t8 t9 t10 t11
   t12 t13 t14 t15 t16] [st]
end

to newmaze
clean
end
```

Instructions are in a text box on a separate page, with a button to go with it. The Instructions page can have a button to go back to the Maze.Maker. Instructions also might have been a pop-up text box, but then I'd have had to add a button to make it go away :

```
to ?
instructions
end
```

References

Bruner, Jerome. (1979). The act of discovery. In *On knowing: Essays for the left hand*. Cambridge, MA: Harvard University Press.

Clayson, James. (1988). *Visual modeling with Logo*. Cambridge, MA: MIT Press.

Eadie Adamson
1199 Park Avenue, Apt. 3A
New York, NY 10128
AppleLink: EadieA

by Harmoney Kaonohi
and Melissa Mench

Using Logo to Teach: The Case of Triangles With Unequal Sides

by Glen L. Bull, Gina L. Bull, and Margie Mason

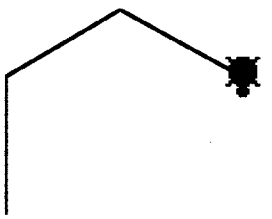
The vision of Logo Seymour Papert outlined in Mindstorms described a world in which students might be engaged in math, science, and language arts through computer explorations. In actual practice, Logo is often used in many classrooms as an electronic etch-a-sketch or as an introduction to programming. There is nothing inherently wrong with this, but there is a significant difference between teaching about Logo and using Logo to teach. In one use Logo is an end in itself, while in the other it is used as lens for exploration of another subject. This distinction may have been a more difficult one than the originators of Logo anticipated, given the limited integration of Logo into the curriculum of most classrooms.

Still, in the hands of a practiced teacher, Logo can be an effective springboard for exploration of math and other content areas. Jim McCauley's article "Kepler" in the December/January, 1983-84, issue of *The Computing Teacher* is still perhaps one of the best descriptions of this process and its potential for establishing a collaborative environment among instructors and learners. We were reminded of this recently when a student in a teacher education class asked a question we had not heard in more than a decade of teaching Logo courses.

The question came about during our standard introduction to the Logo turtle. After an initial review of basic turtle commands, the class is asked to develop a procedure for creating an equilateral triangle. Even experienced math teachers who have not encountered Logo before sometimes fail to recognize that the external rather than the internal angle is required as the input. Therefore, a high percentage of the initial attempts typically consist of a procedure like this:

```
TO Triangle
REPEAT 3 [FORWARD 50 RIGHT 60]
END
```

As moderately experienced Logo users will recognize, this procedure produces the following result:



Some reflection and perhaps a stint of playing turtle usually causes a few in the class to realize that the external angle (120) rather than the internal angle is the required input. After the required angle is established for a triangle of fixed size, it is a short step to introduce a variable triangle:

```
TO Tri :Size
REPEAT 3 [FORWARD :Size RIGHT 120]
END
```

The point of the exercise is two-fold: that Logo can be employed for interesting activities even by novices after only a few minutes of use, and that it has the potential to serve as an "object to think with." It is also worth noting, as Papert does in *Mindstorms*, that the sum of the exterior angles of a polygon always equals 360 degrees. Therefore, in the turtle's geometry the angle needed for creation of any regular polygon can be determined by dividing 360 by the number of sides. For example, the Logo turtle would need to turn 60 degrees six times to draw a hexagon.

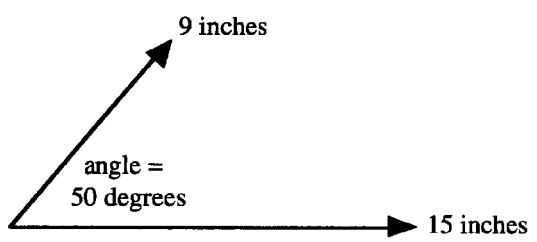
```
TO Hexagon :Size
REPEAT 6 [FORWARD :Size RIGHT 60]
END
```

In this instance one of the members of the class wondered if Logo could be used to draw a triangle with unequal sides. The answer, of course, was "Yes," and this was the result of our first attempt to create a scalene triangle, which included three inputs for specification of the first two angles, and the size of the triangle:

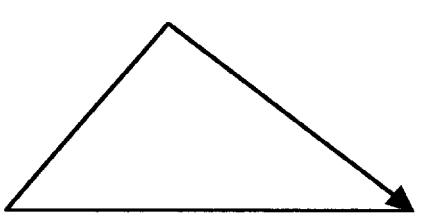
```
TO Scalene :Angle1 :Angle2 :Size
FORWARD :Size
RIGHT :Angle1
FORWARD :Size
RIGHT :Angle2
FORWARD :Size
RIGHT 360 - (:Angle1 + :Angle2)
END
```

You have of course spotted the bug in this procedure, but in the give and take of class we did not recognize the problem until we were surprised by the following result:

It was obvious in retrospect that if a triangle is to have sides of different lengths, the same variable cannot be used for all three sides. Further, some vaguely remembered remnants of high school geometry caused us to recall that two sides and the angle in between are sufficient to determine the dimensions of a triangle. For example, if a triangle has two sides of 9 inches and 15 inches with an angle of 50 degrees between the two, it is clear that the third side and remaining two angles are also determined by these specifications:



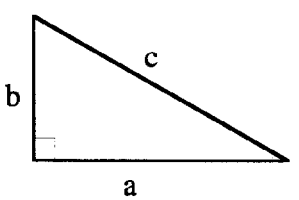
If the two sides of this triangle are drawn on a piece of paper, a pencil can be used to create the third side by drawing a line between the ends of the two existing sides:



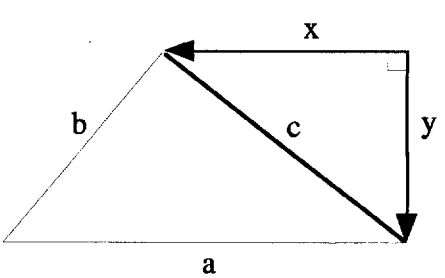
However, in the Logo universe the third side is drawn by telling the turtle how far to go, and in what direction. This kind of conversation with the turtle can generate interesting mathematical explorations. Our first attempt to tell the turtle how far to travel foundered on the fact that we did not know how far it should go:

```
TO Incomplete.Scalene :Angle1 :Side1
   :Side2
FORWARD :Side1
RIGHT :Angle1
FORWARD :Side2
RIGHT ???
FORWARD ???
RIGHT 360 - (:Angle1 + :Angle2)
END
```

A good Logo rule says that "if you are not sure how to proceed, look for a similar problem that you can solve." Fragments of high school math once again came to the rescue. We remembered that the Pythagorean theorem states that if two sides of a right (90 degree) triangle are squared and added together, they will equal the square of the hypotenuse: $a^2 + b^2 = c^2$. Therefore we should be able to calculate the third side of a triangle with a 90 degree angle.

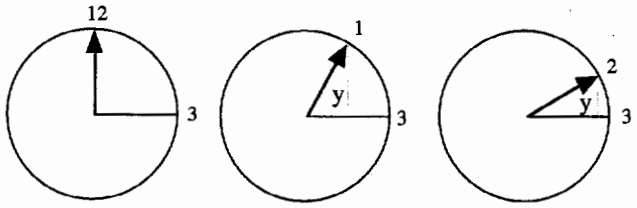


With this in mind, it appeared that if we could find the length of lines x and y in the diagram below, we would be able to determine the third side of our triangle (represented by Line c).



There happens to be a straightforward way to find the height of Line y in the figure above. Imagine a clock with a radius of 1 meter, which is set to 3:00 o'clock. In this case the vertical distance between the minute hand, standing straight up, and the hour hand will be 1 meter. After five minutes goes by, the minute hand will be at 1 on the clock, forming a 60-degree angle with the minute hand. At this point, the height of a line drawn straight down from the tip of the minute hand to the

hour hand will be about .9 meters. (Let's call the line drawn from the tip of the minute hand to the hour hand "y".) If another five minutes goes by, and the minute hand is at 2 on the clock, forming a 30-degree angle with the hour hand, the height of Line y will be .5 meters.



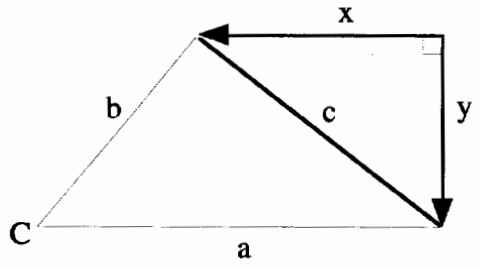
If angle = 90, y = 1 If angle = 90, y = 0.87 If angle = 90, y = 0.5

It is apparent that when the time is 3:15, and the minute hand has moved all the way to the number 3, the distance between the minute hand and the hour hand will be 0. (To simplify matters, we are assuming that the hour hand has not moved at all.) These distances can be verified empirically by drawing a circle on the blackboard (or a large sheet of paper) and measuring the length of Line y for each of the positions shown above.

There is a mathematical function, known as the "sine" function, that describes the distance between the tip of the minute hand and the hour hand (Line y) in the example above. This function is expressed in Logo as the *sin* function. The sin function allows Logo to tell us the height of Line y for any given angle:

```
PRINT SIN 90
1
PRINT SIN 60
.867
PRINT SIN 30
.5
PRINT SIN 0
0
```

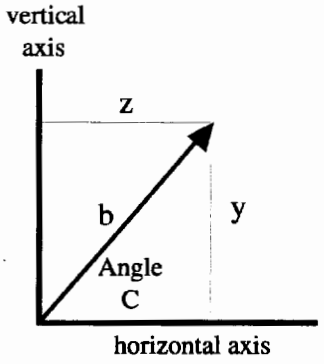
In this example we are assuming that the length of the minute hand is 1 meter. If we have a larger clock, with a 2-meter minute hand, the result would be doubled. In any other case in which the length of the minute hand was not 1 meter, the height of Line y could be found by multiplying the sine of the angle between the minute hand and the hour hand times the length of the minute hand.



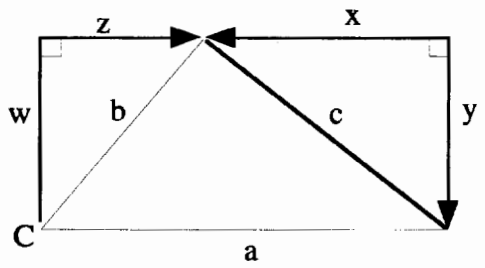
More generally, in our original example, the height of Line y can be found by multiplying the sine of Angle C times the length of Line b (equivalent to the minute hand from the previous illustration). Therefore the height of Line y = Line b * sine Angle C. This can be expressed in Logo as:

```
MAKE "Line.y :Side.b * sin :Angle.C
```

In the same way that sine of Angle C can be used to determine the distance between the tip of the minute hand (Line b) and the horizontal axis, the cosine of Angle C provides the distance between the tip of the minute hand and the vertical axis (Line z in the illustration below).



In the original problem, we know the length of Sides a and b in the scalene triangle below, and are attempting to find the length of Side c. The Pythagorean theorem tells us that $c^2 = x^2 + y^2$, and we have found that y = Side b * sine Angle C.



We still need to find the length of Line x. It appears as though Line x = Line a - Line z. We have just found that Line z = Side b * cosine Angle C. This could be expressed in Logo as:

```
MAKE "Line.z :Side.b * cos :Angle.C
```

With this piece of information, we have everything that we need to know to find the third side of the scalene

triangle (Side c). To find Side c, given the other two sides (Side a and Side b) and Angle C:

```
TO Find.Side.c :Angle.C :Side.a
   :Side.b
MAKE "Line.y :Side.b * sin :Angle.C
MAKE "Line.z :Side.b * cos :Angle.C
MAKE "Line.x :Side.a - :Line.z
MAKE "Side.c SQRT (:Line.x * :Line.x
   + :Line.y * Line.y)
END
```

It has become clear that even a simple Logo problem, such as asking the turtle to draw a triangle with unequal sides, can quickly take us into consideration of intermediate mathematical concepts. We now have all three sides and one angle of our triangle. If we can find a second angle, the third angle can be obtained by subtracting the sum of the first two angles from 180 degrees (since the sum of the interior angles of a triangle equals 180 degrees). Determination of the second angle, given one angle and three sides, can be obtained through use of the arcsine function. The dialect of Logo that we were using did not have an arcsine function, but it did have an arctangent function. The arctangent function can be used to write an arcsine function:

```
TO Arcsin :Number
OUTPUT ARCTAN :Number / (SQRT (1 -
   (:Number * :Number)))
END
```

With the arcsine function in hand, the remaining two angles can be calculated in the following way:

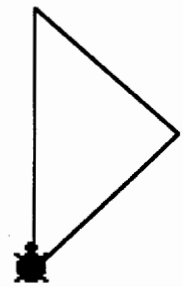
```
TO Find.Angles :Angle.C :Side.a
   :Side.b :Side.c
MAKE "Angle.B (Arcsin ((:Side.b *
   sin :Angle.C) / :Side.c))
MAKE "Angle.A 180 - (:Angle.B +
   :Angle.C)
END
```

We can now write a procedure to draw a scalene triangle when one angle and two sides are known. We wanted to express the angle in terms of the interior angle rather than the exterior angle that the turtle normally uses, so we also created a Right.Turn procedure that accepts the internal angle as an input.

```
TO Scalene :Angle.C :Side.a :Side.b
Find.Side.c :Angle.C :Side.a :Side.b
Find.Angles :Angle.C :Side.a :Side.b
   :Side.c
FORWARD :Side.b Right.Turn :Angle.A
FORWARD :Side.c Right.Turn :Angle.B
FORWARD :Side.a Right.Turn :Angle.C
END
```

```
TO Right.Turn :Internal.Angle
RIGHT 180 - :Internal.Angle
END
```

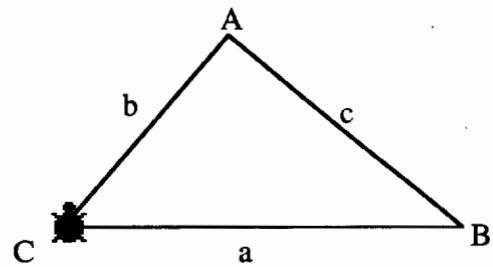
Our second scalene procedure is an improvement over our first attempt described at the beginning of our column. After viewing the result, we decided to make one final refinement.



The turtle starts out facing up as it begins drawing the triangle. However, we decided we would like the base of the triangle to be horizontal rather than slanted. This is accomplished by subtracting Angle A from 90 degrees before the turtle draws the first side of the triangle.

```
TO Scalene :Angle.C :Side.a :Side.b
Find.Side.c :Angle.C :Side.a :Side.b
Find.Angles :Angle.C :Side.a :Side.b
   :Side.c
RIGHT 90 - :Angle.C
FORWARD :Side.b Right.Turn :Angle.A
FORWARD :Side.c Right.Turn :Angle.B
FORWARD :Side.a RIGHT 90
END
```

The revised scalene procedure produced the result shown below. Although the code in the Scalene procedure is less symmetrical, we prefer the visual esthetic of a triangle that has a horizontal rather than a slanted baseline.



Our scalene procedure worked well as long as Angle B was acute, but failed when the angle was obtuse (greater than 90 degrees). To fix this bug, it was necessary to add a middle line to the Find.Angles subprocedure, which substitutes the complement of

Angle B for the result obtained in the first line of the procedure in instances in which Angle B is obtuse.

```
TO Find.Angles
MAKE "Angle.B Arcsin ((:Side.b * sin
   :Angle.C) / :Side.c)
IF :Line.z > :Side.a [MAKE "Angle.B
   180 - :Angle.B]
MAKE "Angle.A 180 - (:Angle.B +
   :Angle.C)
END
```

The final result of our Scalene procedure and associated subprocedures could have been compressed into terser code. For example, in Find.Side.C, we calculated Line z as an intermediate result before obtaining Line x. It would be possible to complete this calculation in a single line without the intermediate step. However, the goal of this programming exercise is not to develop compact code. This program will never be the basis of production code underlying a commercial application such as a spreadsheet or a CAD/CAM program. Therefore, speed and compactness are irrelevant. What is relevant is readability, and the degree to which the *process* of creating the program facilitates exploration of the mathematical concept.

How does exploration of mathematical concepts using Logo differ from traditional approaches to math education, and how can such uses complement the existing curriculum? In our explorations, Logo served as a compass that gave us immediate feedback that allowed us to determine whether our solutions were correct. With traditional pencil-and-paper proofs, it is not always apparent whether a proof is correct. (If it were, students presumably would not submit incorrect proofs.)

Logo is not a substitute for these traditional kinds of mathematical exercises. However, it can provide a useful tool for tinkering, and can be used to shed light on underlying concepts. We moved between a traditional mathematics text and Logo as we were attempting to understand the underlying constructs needed to generate a scalene triangle. The sine function on the pages of the math text is static and unchanging. The same function in Logo is dynamic, and can be used to put the proof to the test. However, by the same token, without underlying mathematical knowledge, the sine function in Logo is useless. Countless students have perhaps briefly examined the sine function in Logo, and then discarded it as an incomprehensible curiosity. The combination of both underlying mathematical knowledge and Logo is needed.

Similarly, a teacher who uses Logo effectively must have an underlying knowledge of both Logo and the content area. Armed with knowledge of both domains, teachers can shift from teaching about Logo to using Logo to teach as well.

Glen Bull is an associate professor in the instructional technology program of the Curry School of Education at the University of Virginia. Gina Bull is a computer systems engineer in the information technology and communication organization at the University of Virginia. By day she works in a UNIX environment, by night in a Logo environment. Margie Mason is a math education professor in the Curry School of Education. Her research interests include educational technologies.

Internet Addresses: GBull@Virginia.edu, Gina@Virginia.edu, MMason@Virginia.edu

Statement of Ownership, Management and Circulation (Required by 39 U.S.C. 3685). 1A. Title of Publication, **Logo Exchange** B. Publication No., 08886970. 2. Date of filing, October 26, 1993. 3. Frequency of issue, quarterly. 3A. No. of issues published annually, 4. B. Annual subscription price, $30 in U.S. 4. Complete mailing address of known office of publication (not printer), ISTE, 1787 Agate St., Eugene, OR 97403-1923. 5. Complete mailing address of the headquarters of general business offices of the publisher (not printer), ISTE, 1787 Agate St., Eugene, OR 97403-1923. 6. Full names and complete mailing addresses of the publisher, editor, and managing editor, Publisher, ISTE; Editor, Sharon Yoder; Managing Editor, David Moursund; 1787 Agate St., Eugene, OR 97403-9905. 7. Owner, International Society for Technology in Education, 1787 Agate St., Eugene, OR 97403-1923. 8. Known bondholders, mortgages, and other security holders owning or holding one percent or more of total amount, None. 9. The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes has not changed during preceding 12 months. 10. Extent and Nature of Circulation. Average No. Copies Each Issue During Preceding 12 Months. A. Total no. copies (net press run), 1,390. B1. Paid Circulation, sales through dealers and carriers, street vendors and counter sales, 0. B2. Paid Circulation, mail subscriptions, 1,200. C. Total Paid Circulation (sum of 10B1 and 10B2), 1,200. D. Free distribution by mail, carrier or other means, samples, complimentary, and other free copies, 120. E. Total Distribution (Sum of C and D), 1,320. F1. Copies not distributed, office use, left over, unaccounted, spoiled after printing, 70. F2. Copies not distributed, returns from news agents, 0. G. Total (Sum of E, F1 and 2—should be equal to net press run shown in A), 1,390. Actual No. Copies of Single Issue Published Nearest to Filing Date. A. Total no. copies (net press run): 1,375. B1. Paid Circulation, sales through dealers and carriers, street vendors and counter sales, 0. B2. Paid Circulation, mail subscriptions, 1,133. C. Total Paid Circulation (sum of 10B1 and 10B2), 1,133. D. Free distribution by mail, carrier or other means, samples, complimentary, and other free copies, 79. E. Total Distribution (Sum of C and D), 1,212. F1. Copies not distributed, office use, left over, unaccounted, spoiled after printing, 163. F2. Copies not distributed, returns from news agents, 0. G. Total (Sum of E, F1 and 2—should be equal to net press run shown in A), 1,375. 11. I certify that the statements made by me above are correct and complete. Patti Van Ordstrand, Distribution/Circulation Manager, ISTE.

ISTE PUBLICATIONS

PRESENTS NEW BOOKS AVAILABLE FROM ISTE

Math Activities Using LogoWriter—Patterns and Designs

by Gary Flewelling

Includes programs that use stamp and sticker designs to explore color patterns, circular fraction paths, tartan design, tessellation, fractals, angles, and so on. Grade levels for the activities range from 1-12 (ages 6-18).

Math Activities Using LogoWriter—Numbers and Operations

by Gary Flewelling

Includes programs and activities for exploring number concepts using electronic counters, Cuisenaire rods, sets of rectangles, and number boards; addition and subtraction using electronic versions of base-10 blocks and colored blocks; and approximate numbers. Grade levels range from 1 to 10 (ages 6 to 16).

Math Activities Using LogoWriter—Investigations

by Gary Flewelling

Includes programs that explore creating geometrical designs by connecting dots in an array, spirolatera designs, digital root patterns, prime numbers, repeating decimals, and perpendicular lines. Grade levels range from 4-12 (ages 9-18).

ISTE DISTRIBUTED

PRESENTS NEW BOOKS AVAILABLE FROM ISTE

Logo for the Macintosh—An Introduction Through Object Logo With the Student Edition of Object Logo

by Harold Abelson & Amanda Abelson

This book is a comprehensive tutorial that teaches both beginning and advanced concepts of Logo programming to beginning programmers, ages 11 and up. Topics covered include: turtle graphics, procedures, debugging, variables, object-oriented programming, and interactive programming. The Student Edition has almost all of the features of the full version, except the Robotics and MIDI modules and the file compiler and application generator. Grades 6-12 (ages 11-18).

The Children's Machine—Rethinking School in the Age of the Computer

by Seymour Papert

Seymour Papert explores what has happened to the computer revolution in education and examines how we should rethink education and let students become creators of their own knowledge. He examines how technology can help students acquire knowledge in much the same manner that infants learn about their world—through exploration.

Object Logo for the Macintosh

Object Logo 2.61 (Macintosh)

Object Logo is an advanced extension of the Logo language. Features include: object-oriented programming; AppleLogo™ compatibility; advanced numeric, symbolic, and list processing; turtle graphics with multiple object-oriented turtles; and MIDI, MacinTalk, and 32-bit color QuickDraw support. The price includes the book *Logo for the Macintosh—An Introduction Through Object Logo.*

Using *LogoWriter* to Tune Turtles and Make Sweet Music

by John Gough

One of the most attractive features of LogoWriter (and other dialects of Logo) is the ability to make music. Many beginners are visually encouraged by seeing a turtle dragging lines around the screen, but hearing musical notes can be simply breathtaking. And creating songs is an extremely simple and direct way to introduce the concept of "procedures" and techniques for writing and editing these procedures.

However, there can be problems. Using the standard information about making turtle music, listeners rapidly discover that the higher the notes go, the more some kinds of computers have tin ears. Listening to a turtle singing flat can be very discouraging, and unpleasant. This article suggests ways to eliminate the problem. It teaches the turtle to sing in tune and introduces some musical procedures that show a little of what you can do with music in Logo.

Starting to Make a Tune

The *LogoWriter* primitive

```
TONE frequency duration
```

plays a note of the specified frequency (or pitch) that lasts for a given duration. (Check your *LogoWriter* documentation for the length of the duration.) The command TONE will not work if it is not followed by two appropriate numbers.

You can type several TONE commands on a single line in the Command Center.

```
TONE 200 20 TONE 400 20 TONE 800 20 TONE
   600 30 TONE 300 30
```

It's not a great tune, but it's a start. Of course, you can write procedures to play music. Go to the Flip side of a *LogoWriter* "page" and type the following:

```
TO LADDER
TONE 200 20
TONE 400 20
TONE 800 20
TONE 1600 20
TONE 1200 30
TONE 600 30
TONE 300 30
TONE 150 30
END
```

Then flip to the Front of the Page and in the Command Center type LADDER and then press Return. The notes you hear are the separate tone commands you have strung together into a procedure or little program called LADDER.

This example is not very musical. The tune called LADDER is more of a mathematical pattern than a song. To make it become musical, it is necessary to know how to make the computer play notes like a musical instrument.

Here is another example. This time it is a procedure called BEETHOVEN.5. Notice that it is easy to introduce the idea of subprocedures when working with music. Type BEETHOVEN.5 in the Command Center to hear the tune.

```
TO BEETHOVEN.5
PHRASE.1
PHRASE.2
END

TO PHRASE.1
REPEAT 3 [TONE  392 5]
TONE  311 20
END

TO PHRASE.2
REPEAT 3 [TONE 349 5]
TONE 294 20
END
```

This procedure is more recognizably tuneful. The numbers are not simple multiples of 100. For students to go further in building musical procedures, they need to have a chart of numbers to get the pitch and the duration correct. This could be an appropriate time to teach some simple music staff reading and simple rhythm notation, perhaps with reference to a piano keyboard or a xylophone or other simple musical instrument. The chart in the next section gives all the necessary information. Even students who have never previously read a note of music can take a piece of written music and use the table as a coding exercise to construct a procedure that will play the music. Of course only one note at a time can be played. *LogoWriter* can play a melody, but it cannot play a chord.

Making the Turtle Musical

In the following table, each number is a value of the *frequency*. The smaller the frequency number, the lower the note; the higher the frequency, the higher the note. These notes go from deep and growly to high and squeaky. (Check the documentation for your version of *LogoWriter* to determine the lowest and highest numbers you can use for the pitch.)

The frequencies given in the table should be thought of as a piano keyboard cut up into sections, so that the B of 123 is followed immediately above by C of 131, and so on. The three middle columns give the nicest sounding musical range.

Note that the values of m have all been corrected so that the notes do not get flatter as they go higher!

Note	Frequency by Octave				
B	123	247	494	1030	2350
B flat or A sharp	117	233	466	980	2160
A	110	220	440	914	2023
G sharp	104	208	415	880	1955
G	98	196	392	804	1754
F sharp	92	185	370	765	1670
F	87	175	349	718	1561
E	82	165	330	675	1450
D sharp or E flat	78	156	311	647	1400
D	73	147	294	603	1286
C sharp	69	139	277	569	1220
C	65	131	262	532	1114

Special Comment: Middle C = 262

You might try these suggestions for the value for the duration:

```
Whole note      40
Half note       20
Quarter note    10
Eighth note     5
```

To make a pause between notes, use

```
tone 0 duration
```

If your version of *LogoWriter* won't accept an input of 0 for TONE, then you can use the command WAIT to insert pauses in your music. (Check your *LogoWriter* documentation for details about the input to WAIT.)

Tuning the Turtle

When I began using *LogoWriter*, I was delighted to be able to make it play tunes, but disturbed that the notes got flatter the higher the numbers went. I had noticed something similar with BASIC music, and thought that this was just a computing glitch that no one had solved. However, I later tried some simple experiments with tone that showed me that the fault was really with the values in the manual, not with the computer or the language.

First I found that, apparently contrary to the laws of acoustics, simply doubling the frequency of a note does not, with *LogoWriter* at least, always lead to the next octave of the note. The procedure CEE.OCTAVES shows how "doubling" gets flatter and flatter as the numbers get larger.

```
TO CEE.OCTAVES
TONE 65 20
TONE  131 20
TONE  262 20
TONE  524 20
TONE  1048 20
TONE  2096 20
TONE  4192 20
TONE  8384 20
END
```

Next I wanted to find out whether TONE responded only to the values for frequency given in the manual. I had initially experienced some difficulties with values around 1000, and believed that TONE stopped working in any reasonable way beyond about 1000. So I made a procedure that tested every possible whole number value for m from any given starting point. It was not pleasant listening, but it answered my questions about what possible values of m could be used.

```
TO SCALE.UP :FREQUENCY
IF :FREQUENCY > 9000 [PRINT [Too
   squeaky for me!] STOP]
TONE :FREQUENCY 1
SCALE.UP :FREQUENCY + 1
END
```

If you can't stand the sound, you can always use the stop keys!

It is interesting that even though the values of the frequency are steadily increasing in SCALE.UP, the notes do not always sound as though they are going smoothly up the musical scale, like a violin player sliding his finger along the string as he runs the bow over the string. Instead it sounds as though there are sometimes jumps from one note to another, like the frets on a guitar's fingerboard.

Next I wanted to see what would happen as I moved down the scale. I adapted the SCALE.UP procedure to move down. Watch for an error message when you run this as the value of the frequency gets too small.

```
TO SCALE.DOWN :FREQUENCY
IF :FREQUENCY < -100 [PRINT [That's
   too deep for me] STOP]
TONE :FREQUENCY 10
PRINT :FREQUENCY
WAIT 5
SCALE.DOWN :FREQUENCY-1
END
```

In fact, TONE doesn't work with negative numbers, so the second line, which is meant to control the recursively decreasing values of FREQUENCY, never runs. What is the lowest value of FREQUENCY you can hear?

Then I needed to find the values of FREQUENCY that would give me notes that would be in tune. You can hear that the low values given in the manual are reasonably in tune. But I needed to tune the higher values. Because the higher notes were flat, I knew I would have to add some extra amount to FREQUENCY so that the note would sound right.

This next procedure lets you do what a piano tuner does when one note is matched against another until the two notes are an octave apart. Starting with any value of FREQUENCY, it plays the corresponding note and then plays a higher note that has a frequency that is twice as great plus an extra bit that you also specify. It also prints the values it is running.

```
TO TUNE.OCTAVE :FREQUENCY :EXTRA
TONE :FREQUENCY 40
TONE :EXTRA + 2 * :FREQUENCY 40
(PRINT :FREQUENCY [and] 2 *
   :FREQUENCY [+] :EXTRA [ = ] 2 *
   :FREQUENCY + :EXTRA)
END
```

Experiment with

```
TUNE.OCTAVE 700
```

and some different values for EXTRA until you think it sounds like an octave interval between the two notes. This is the procedure I used to obtain the corrected values for m that are given in the table.

Using Nonmusical Sound Effects

It is not necessary to stick with musical sounds. The primitive TONE can also be used to make sound effects. These can be built into larger procedures, such as those for animation or story telling. The primitive term TONE can be used to dress up a little cartoon or piece of animated graphics, which is one of the most exciting things beginners can learn to do with *LogoWriter*. Here are some simple samples.

```
TO STEP
TELL 1
PENUP
SETHEADING 270
SETSH 16
ST
TONE 50 1
HIDE TURTLE
FORWARD 5
SETSH 17
ST
TONE 60 1
END

TO WALK
REPEAT 10 [STEP]
END
```

Of course TONE 50 1 and TONE 60 1 are actually musical notes, although they are not in tune with the scale given in the table. But because they are so short, they sound like clicks as the little man walks from right to left.

Here are a few more simple procedures that demonstrate interesting sounds. They could be used before any tuned musical work to explore procedures, experiment, stimulate problem solving, and get students involved in working with Logo without going down the over-trodden path of turtle graphics.

```
TO SPACE.NOISE
REPEAT 30 [TONE 20 + RANDOM 9000 1 +
   RANDOM 15]
END
TO RAIN.NOISE
REPEAT 30 [TONE 500 + RANDOM 1000 1
   WAIT 1 + RANDOM 10]
END

TO SIREN :NUMBER :STEP
IF :NUMBER > 6000 [STOP]
TONE :NUMBER 1
SIREN :NUMBER + 10 * :STEP :STEP *
   1.5
END

TO CHIRP :TIMES
REPEAT :TIMES [SIREN 1000 10]
END
```

You might challenge students to make the sounds of a squeaky door, a dentist's drill, a motorbike engine revving, a dripping tap, a croaking frog, a bird whistling, a phone ringing, or a ball bouncing with the bounces getting lower and shorter in time. The possibilities are endless. Of course there are limitations. *LogoWriter* can play only one note at a time. There is always a tiny pause between each note. But it's fun.

Making the Turtle Sing a Song With Words

A musical tune can be enhanced by making a picture appear on the screen while a tune plays. If you have the tune for "Mary Had a Little Lamb," then you might make a lamb shape and then make a procedure that will bounce the lamb across the screen while the music plays, note by note. To accomplish this, put some commands together that will show the lamb-turtle, play a note, move the lamb, play a note, move the lamb, and so forth.

You can also make the words appear as the music plays.

```
TO SING.WORD
PRINT [Baa]
TONE 262 20
PRINT [Baa]
TONE 262 20
PRINT [Black]
TONE 392 20
PRINT [Sheep]
TONE 392 20
PRINT [Have]
TONE 440 10
PRINT [you]
TONE 494 10
PRINT [a-]
TONE 532 10
PRINT [-ny]
TONE 440 10
PRINT [wool?]
TONE 392 40
END
```

You can also use INSERT to run the words along a line instead of each word being on a successive line.

```
TO SING.WORD.1
PRINT [Baa]
TONE 262 20
CB
(INSERT [ ] [Baa])
TONE 262 20
(INSERT [ ] [Black])
TONE 392 20
(INSERT [ ] [sheep])
TONE 392 20
END
```

Similarly, LABEL can be used to make words seems to come from the mouth of a turtle shape or a character drawn using turtle graphics. An alternative way of linking music, words, and action displays a line of words and makes a ball or some other turtle shape bounce from word to word as the music plays. Can you do it?

Here is an example that combines random music and pictures, a kind of scribbling kaleidoscope, or singing Brownian motion.

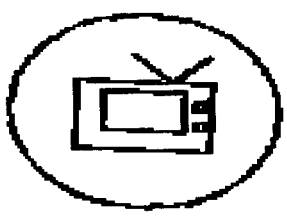
```
TO SING.KITTEN
PU
SETC 2
SETSH 22
REPEAT 10 [TONE 200 + RANDOM 790 1 +
   RANDOM 20 FORWARD RANDOM 20 RIGHT
   TURN RANDOM 360]
END
```

It should be clear that TONE gives a quick way to start students programming. For anyone who has only seen turtle graphics, this should be arrestingly different and fun—an alternative first door into the world of Logo. By making a few adjustments, almost all of the above procedures should work in any Logo dialect that has a primitive that sings. Try it.

And, lo, the voice of the turtle was heard throughout the land!

John Gough has worked in tertiary teacher education for more than 15 years. He is the author of *Learning to Be a LogoWriter Writer: A Resource Manual For Adult Learners* (Deakin University Press), a project that grew from nearly five years experience teaching *LogoWriter*. He is interested in *LogoWriter* applications for school mathematics and is currently working on a resource manual and teaching stacks for *HyperCard* and its Logo-like programming language, HyperTalk.

John Gough
Lecturer in Mathematics Education
Deakin University (Toorak Campus)
PO Box 224
Malvern
Victoria 3144



by Kelly Cripps and
Dane MaCullam

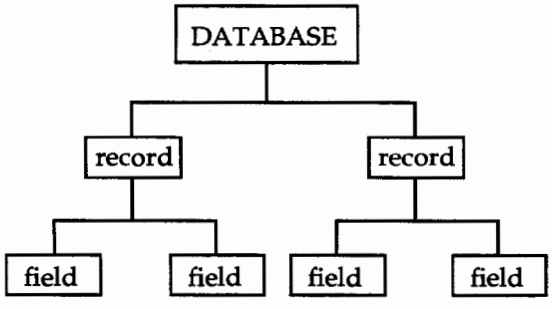
Hypermedia Design—Dynamic Information Management

by Jandy Bird

It is widely acknowledged that a primary task of education today is to help children become effective users and manipulators of information, which is increasing in exponential leaps and bounds. Hypermedia tools provide students with an effective, challenging, and rewarding way to use information productively.

What is hypermedia? I see hypermedia as a way to use the computer to link information in a relational way. The hypertext idea goes back as far as to 1945 (Adamson, 1991). It is only recently, however, that this technology has become readily available to most of us. Eadie Adamson (1989) and Michael Tempel developed *LogoWriter Hypermedia Tool* using *LogoWriter,* which recently became available from LCSI ($39.95). This tool enables students to turn a word on a *LogoWriter* page into a "button" that automatically links to another page (or to a videodisc player if you have one). When we received this tool, I demonstrated it to my third- and fourth-grade enrichment resource room students, who were instantly fascinated, even without a videodisc component!

Prior to hypermedia, a database was a commonly used way to store and manage information, and the information was stored and retrieved in a hierarchical manner. Each *record* had a series of *fields,* and all the records together made the database.



Organizing data using a database meant organizing it in a hierarchical fashion. A database on the U.S. Presidency, then, meant that for every President the student had to first decide what fields should be used: birth and death dates, place of birth, wars fought during the term of office, and so forth. Then the student had to find those particular facts for each President.

Hypermedia allows more flexibility in organizing data. The information is assembled on a *card* (in this case, a *LogoWriter* page), and the user can "click" on a "button" to find information of choice. Glen and Gina Bull (1990) point out that the *LogoWriter* metaphor of a book with pages is "well suited to a hypermedia environment" (p. 22). In a hypermedia presentation on the U.S. Presidents, there might be a *card* (scrapbook page) on George Washington. The user might click on the *button word* "Revolution" to see a timeline on Washington's campaigns during the Revolutionary War. "Vernon" might present a graphic illustration of Washington's home, and "Martha" might take the user to a page about Mrs. Washington and her family. The *Hypermedia Tool* automatically capitalizes the words that are buttons and takes the user to that page.

There are two obvious advantages of hypermedia organization. First, it presents a greater variety of kinds of information. Just using *LogoWriter,* the user can employ graphics, text, and interactive programming to involve the user. (*Hypermedia Tool* allows for interface with laser videodiscs, and imported scanned images can also be used.) And this points to the second value of hypermedia, which is that the user can tailor the program to his/her own interests and needs.

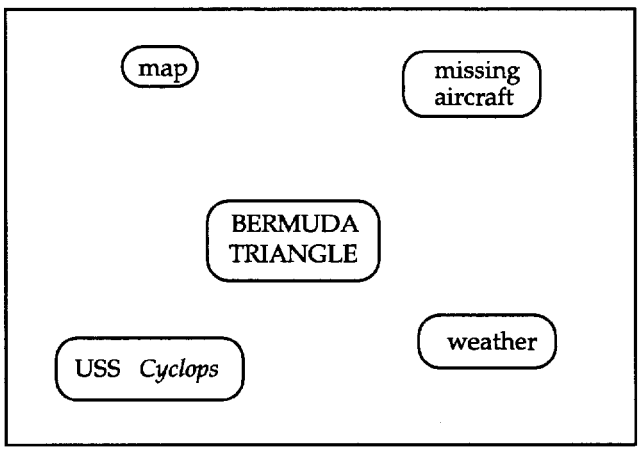
Several of my students were immediately eager to use the hypermedia tool to develop projects in different areas. All of them were familiar with *LogoWriter,* having worked with it in our computer lab and in my resource room for two or three years. Some of the topics chosen for hypermedia presentations were sharks, gravity, and the Bermuda Triangle.

What became immediately obvious was the problem of design. The students gathered different information on their topics and brainstormed graphic and interactive ideas to add to their presentations. Then what? How could this data be organized effectively into a hypermedia presentation?

One of my third graders, who thinks in a very sequential way, began an outline. Our first question was: How do we translate the outline of information

into cards with buttons? The point was not to have one thing inevitably lead to another in strict linear fashion. The process of outlining works well for database organization, but it wasn't so helpful for hypermedia.

Then an idea appeared ... What about a web? Our students were familiar with webbing from our school's process approach to writing. Therefore, each student organized the information in "clusters," with the main topic in the middle. It began to look like this:



After the clusters began to take shape, relational lines could be drawn between idea groups. This defined the "buttons" that would take the user from one place to the next. For instance, on the starting page (home card) the word "map" would become a button to take a curious user to a map of the location of the Bermuda Triangle.

Of course, this led to other considerations. If the user chose to go to the "map" page, what next? Should there be a choice to return to the home card/page or to go to another map? Should there be an automatic return after the map is shown for a certain amount of time? The designer had to make these decisions and program them appropriately.

As the students were developing their hypermedia projects, the webs evolved. It was easy to keep an ongoing record of information to be presented by adding it to the web. Subtopics could be developed during the process, too. For instance, the "weather" page, which was a text discussion, developed an animated illustration of an airplane being hit by lightning. Also, some students found it helpful to put the word "text" or "graphic" or "interactive" right on the web to remind themselves of the kind of information the page presented.

When the students were ready to actually create their presentations on the computer, they had their webs as a guide. As they did each page—represented by a circle on the web—they crossed it out so that their progress was instantly visible. The annotations of "graphic" or "interactive" on the webs were reminders to me for specific individual or small-group lessons on *LogoWriter* commands such as WAIT or READLIST; and if there were related examples to show the students, I could have them ready as each student came to that part of the web.

The students had the web as their plan, but they did not link the button words until *all* the pages were in place and working properly. Just before linking pages, the students printed each page for final editing with the button words circled in colored marker. Again, the students referred to the web for the links. This last-step linking is suggested in the support materials that come with the *Hypermedia Tool* and is important because once you link something, it is hard to unlink it later if you change your mind. The *Hypermedia Tool* allows the students to link all the pages with simple key presses.

As the students completed their hypermedia presentations, they took them back to their classrooms to share with others. They found out that it is more effective to let others use hypermedia individually or in pairs than in a larger group, because a group often disagrees about what to do and only one or two do the work if the group is large. My students were delighted with the flexibility of this form of managing information, and they all had ideas of how to go even further had time permitted.

A note should be made about assessment. Glen and Gina Bull (1992) point out that in the development of a hypermedia microworld students "learn a great deal about the content which they are studying. The process of creating a page on Jupiter may require a trip to the library to find out more about this giant planet" (p. 28). The hypermedia presentation lends itself to performance assessment of the student's understanding of the topic at hand, and provides the teacher with a clear picture of how the youngster has comprehended the concepts. Of course, students and teachers can also evaluate the presentation's effectiveness in addition to this.

Finally, I would note that this was my introduction to using hypermedia, and it was a very exciting one. I have subsequently begun to work with *HyperCard* on the Macintosh, and I am finding that the same principles of design seem to hold true. It still seems like a good idea to plan in a web and to do the linking at the end because getting things out of order is a problem. The *LogoWriter* concept of a page with procedures on the back is similar to a *HyperCard* stack with scripting on each card. In short, the *Hypermedia Tool* allows students to use very powerful technology in a familiar environment and to manipulate and manage information in a creative, exciting, and effective way.

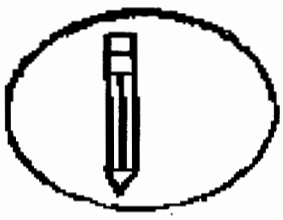
References

Adamson, E. (1989). Hypertext in LogoWriter. *Logo Exchange*, 7(8), 4-6.

Adamson, E. (1991). Logo ideas—Hypertext revisited. *Logo Exchange*, 10(1), 20-23.

Bull, Glen L., Bull, Gina L., and Cochran, Paula. (1990). Creating a hypermedia adventure story with Logo. *Logo Exchange*, 9(2), 23-25.

Bull, Glenn L., and Bull, Gina L. (1992). Hypermedia links with Logo. *Logo Exchange*, *10*(3), 28-32.



by Danielle Wilson
and Rhonda Wingard

Jan J. Bird, Ed.D.
Conover Road School
Colts Neck, NJ 07722
908/946-8590
CIS 73517,3270

Hands-On?

by Douglas H. Clements and Julie S. Meredith*

Is it easier to learn new software by reading the manual first and then trying it on the computer? Or is it better to sit down at the computer with the manual and try things out while you read? The answer may not be obvious.

Hands-On Experience: The Down Side

We started thinking about the question after reading research conducted in Australia by John Sweller and Paul Chandler (in press). Sweller's research has convinced him that writers often mistakenly make their materials "hard to learn from."

Some materials lead to the *split-attention effect.* Learners find it difficult to split their attention between information from two different sources, each of which gives only half the story. The *redundancy effect* is related: The two sources present the same information, but learners still have to attend to all this redundant material, so they learn less.

The researchers had three groups of adult learners learn a word processor: One learned from a manual without pictures plus a computer (the normal methods for this software—the split attention effect), one used a manual with pictures and a computer (the redundancy effect), and one learned from just the manual with pictures. Consistent with their theory, the last group—the one with the *least* hands-on experience—learned the best.

But teachers love hands-on work. Would they experience the same effects?

Teachers Learning With and Without Hands-On Computer Experience

Over the summer we had teachers work on Logo software at the university. Although some of the teachers were a bit familiar with Logo or *Geo-Logo*, all felt they needed another day to focus on *Geo-Logo*. We divided the teachers into two groups. The first group ("computer-and-manual") stayed in the room with the computers and were told to work through the manual on *Geo-Logo*. The second group ("manual") went to a quiet room without any computers to read the manual.

The teachers in the "manual" group were not too happy with the situation. Every teacher expressed a preference to have the computer with them to "try things out." Only one teacher, Terry, said that she usually learned by reading and thought that not having the computer "wasn't that bad."

After two hours, we brought the "manual" teachers back. They had time to work on the computers at their own pace. After another couple of hours, we surveyed all the teachers on their feelings about the experience and about specific *Geo-Logo* features. After lunch, all the teachers worked on a project.

The results: The teachers in the computer-and-manual group felt they had had the better learning experience. However, the observations and survey results showed that this was not necessarily the case. The teachers in the "manual" group were more familiar with the manual and referred to the manual during the project work. Thus, they were capable of completing the project with less help. More than one teacher admitted that they never would have read the manual otherwise. Terry independently completed the project and went on to design another project. Terry had never worked on Logo before.

The study wasn't perfect. Technology problems caused a rather slow start. The room where the teachers read the manuals was small. When a teacher had questions, the discussion bothered some of the other teachers.

Although all the teachers left with the manual and a disk, it was our opinion that the teachers in the "manual" group would use *Geo-Logo* more at home. We plan to investigate this. The issue is important because the teachers were only beginning to explore Logo's possibilities.

Final Words

Although reading a well-designed manual may not seem to be the ideal learning situation, it may be a viable (perhaps even better) way to learn about software. Most classroom teachers do not have unlimited access to computers. Long introductions to new software can limit the time the students have to interact with the software at a deeper level. To make the most of the time on-computer, teachers may want to consider off-computer tasks to acquaint students with software.

The experience leads us to wonder if there isn't a lesson for our teaching too. First, (mindless) hands-on exploration probably doesn't help K-12 students learn Logo either. Second, students may be able to maximize on-computer time—and learning—by both learning

about Logo and planning their own Logo activities off the computer. Balance is critical.

Reference

Sweller, J., & Chandler, P. (in press). Why some material is difficult to learn. *Cognition and Instruction.*

* Time to prepare this material was partially provided by the National Science Foundation under Grants No. MDR-8954664 and MDR-9050210. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Douglas H. Clements, professor at the State University of New York at Buffalo, has studied the use of Logo environments in developing children's creative, mathematics, metacognitive, problem-solving, and social abilities. His book *Computers in Elementary Mathematics Education* emphasizes Logo (Prentice–Hall, 1989). Through a National Science Foundation (NSF) grant, he has developed a K-6 elementary geometry curriculum, *Logo Geometry* (published by Silver Burdett & Ginn, 1991). He is currently working with several colleagues on a second NSF-funded project, Investigations in Number, Data, and Space, to develop a full K-6 mathematics curriculum featuring Logo.

Julie Sarama Meredith is a mathematics education doctoral student at the State University of New York at Buffalo. She has taught secondary mathematics and computer science, gifted math at the middle school level, and mathematics methods courses. Along with Clements, she is currently designing and programming a new version of Logo for the NSF-funded Investigations project.

Douglas H. Clements and Julie Meredith
State University of New York at Buffalo
Department of Learning and Instruction
593 Baldy Hall
Buffalo, NY 14260

CIS: 76136,2027
BITNET: CLEMENTS@UBVMS.CC.BUFFALO.EDU

The Turtle Takes a Cab

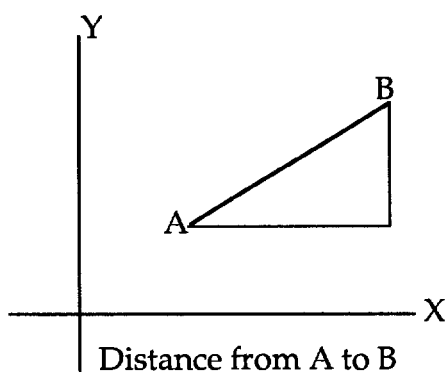
by Kara Ryan, SSND, and Ralph Olliges

The geometry typically taught in schools is of a Euclidean variety, with varying degrees of formalism and rigor. But city students experience a different geometry as they attempt to negotiate the streets and avenues in their neighborhoods. Taxicab geometry, a non-Euclidean geometry, may be more closely related to the lives of our students and hence is worthy of study. Of course, Logo provides an excellent computer environment that can model the pathways students take in the daily travels of their lives. In the following article, Kara Ryan and Ralph Olliges outline how Taxicab geometry can be approached with public school learners. Perhaps you too will find it challenging to ask about the shortest distance between two points, say, or about what are and what are not equivalent paths between two or more points. Read on!

by A. J. (Sandy) Dawson

Taxicab geometry is a non-Euclidean geometry begun in 1952 by Karl Menger. It is unlike the geometries of Bolyai and Lobachevsky. The way distance is measured is changed, which yields some fascinating results. Logo is an ideal tool for exploring Taxicab geometry, because Euclidean geometry and Taxicab geometry both use the same coordinate system.

Two points A and B are given with A = (3,2) and B = (7,5).



In Euclidean geometry, the distance between any two points is

$$d_e(X, Y) = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$$

for $X = (X_1, X_2)$ and $Y = (Y_1, Y_2)$.

In this example, the Euclidean distance between A and B is 5 units. Let us measure the distance from A to B in Taxicab geometry. The general formula for measuring distance in Taxicab geometry is

$$d_e(X, Y) = |X_1 - X_2| + |Y_1 - Y_2|$$

Hence, the distance from A to B is 7 units in Taxicab geometry. Because we will be using Logo to draw in this geometry, imagine a city where all streets are either North-South or East-West. All city blocks are exactly one unit long. If the Logo turtle takes a cab to ride from point A to point B, the turtle will have to pay for a ride of 7 blocks. Of course, the cab must travel on the streets and cannot take a *short cut* through the buildings, as is done in Euclidean geometry. Obviously, there is more than one way to travel from point A to point B, but the distance in each case will be the same, i.e., 7 units.

The Logo procedure TAXIDIST will allow the user to input the coordinates of any two points. The procedure will then draw the line segment AB and calculate the Taxicab distance from point A to point B. This distance will be displayed on the screen. The procedures given here are for the Apple Logo II version of Logo.

```
TO TAXIDIST
CS
COORD
PU
SETPOS (LIST :A1 :A2)
PD
SETPC 3
SETPOS (LIST :B1 :B2)      ..
HT
PRINT "
PRINT "
PRINT (SENTENCE [TAXI DISTANCE = ]
   (:BA1 + :BA2))
END
```

This procedure will call the procedure COORD,

which in turn will call the two procedures READNUMBER and ABS. These two procedures are given here.

```
TO COORD
PRINT (SENTENCE [INPUT X-COORDINATE
   OF POINT A]) MAKE "A1 READNUMBER
PRINT (SENTENCE [INPUT Y-COORDINATE
   OF POINT A]) MAKE "A2 READNUMBER
PRINT (SENTENCE [INPUT X-COORDINATE
   OF POINT B]) MAKE "B1 READNUMBER
PRINT (SENTENCE [INPUT Y-COORDINATE
   OF POINT B]) MAKE "B2 READNUMBER
MAKE "BA1 ABS(:B1 - :AI)
MAKE "BA2 ABS(:B2 - :A2)
END

TO READNUMBER
MAKE "IN FIRST READLIST
TEST NUMBERP :IN
IFTRUE [OUTPUT :IN]
IFFALSE [PRINT [PLEASE ANSWER WITH A
   NUMBER.]]
IFFALSE [OUTPUT READNUMBER]
END

TO ABS :VALUE
IF :VALUE < 0 [OUTPUT - :VALUE]
OUTPUT :VALUE
END
```

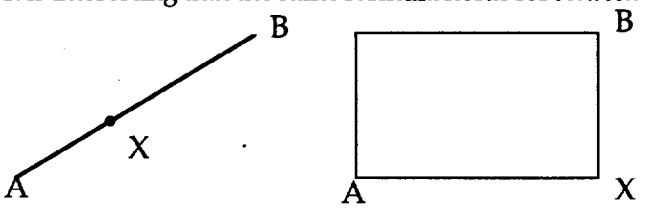
The word *between* takes on an interesting interpretation in Taxicab geometry. In Euclidean geometry, we say that a point C is between points A and B if C is a point on the line segment AB such that

$$d_e(A, C) + d_e(C, B) = d_e(A, B)$$

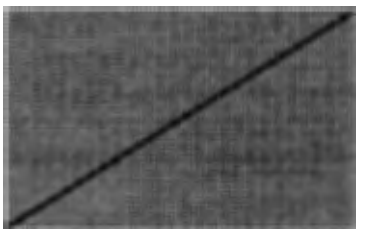
A similar formula applies in Taxicab geometry.

$$d_t(A, C) + d_t(C, B) = d_t(A, B)$$

It is interesting that the same formula holds for *between* in both geometries. In Euclidean geometry, the point X will be a point on the line segment AB, while in Taxicab geometry, X will be any point in a rectangle with A and B as opposite vertices. If the two given points are on a horizontal or a vertical line, then there is no *between* rectangle in Taxicab geometry. In this case, the Taxicab and the Euclidean geometries have identical *between* segments for the two given points. The Logo procedure TAXIBETWEEN for any two given points will draw the *between* rectangle and shade it. Note that the procedure TAXIBETWEEN will call three of the procedures also used by the procedure TAXIDIST, namely, COORD, READNUMBER and ABS. In addition, the procedure SHADE will be called by the TAXIBETWEEN procedure. If you run the procedure TAXIBETWEEN using the coordinates A = (-60, -30) and B = (70, 50), you see:



"Betweenness" in two geometries



The Taxicab distance from A to B is 210.

```
TO TAXIBETWEEN
CS
COORD
PU
SETPOS (LIST :A1 :A2)
PD
SETPC 2
SHADE (2 * :BA1):BA2
PU
SETPOS (LIST :A1 :A2)
PD
SETPC 3
SETPOS (LIST :B1 :B2)
HT
PRINT "
PRINT "
PRINT (SENTENCE [TAXI DISTANCE =]
   (:BA1 + :BA2))
END

TO SHADE :BA1 :BA2
REPEAT :BA1 [FORWARD :BA2 BACK :BA2
   RIGHT 90 FORWARD .5 LEFT 90]
END
```

Conclusion

Betweenness is just one of the many interesting results obtained in Taxicab geometry. This geometry is simple to understand, making it an ideal means to study a non-Euclidean geometry. Logo supplies an effortless method to observe some of the fascinating results of Taxicab geometry. The fact that Euclidean geometry and Taxicab geometry both use the same coordinate plane makes Logo an ideal instrument to study Taxicab geometry. Any geometric concept using distance will create interesting effects in Taxicab geometry. Hopefully, using the powerful tool of Logo with Taxicab geometry will stretch your students' minds to new boundaries.

Kara Ryan has a doctoral degree in mathematics from Saint Louis University. She currently is the department chairperson of the Mathematics/Computer Science Department at the College of Notre Dame of Maryland. She teaches mathematics courses and does research in the areas of finite geometries and Taxicab geometry.

Dr. Kara Ryan
Mathematics & Computer Science Department
College of Notre Dame of Maryland
4701 N. Charles Street
Baltimore, MD 21210

Ralph Olliges has a doctoral degree in education with a computer emphasis from Saint Louis University. He currently works as a systems analyst in the computer center at Parks College. He also teaches mathematics and computer science courses at Saint Louis University.

Dr. Ralph Olliges
Computer Center
Parks College of Saint Louis University
500 Falling Springs Road
Cahokia, IL 62206
olligesrh@sluvca.slu.edu

A. J. (Sandy) Dawson
Faculty of Education
Simon Fraser University

by Kelly Cripps and
Dane McCallum

ISTE Books & Courseware Order Form

To order ISTE products advertised in this publication, find the product title in the following list and enter it on the form below.
To receive a free catalog with a complete listing of ISTE products and services, please call our toll-free number.

Product	*Nonmember Price*	*Member Price*
Internet Companion—A Beginner's Guide to Global Networking	12.95	12.95
The Whole Internet—User's Guide & Catalog	24.95	24.95
• ClassWorks: AppleWorks for the Classroom	54.95	45.95
• ClassWorks: Microsoft Works for the Classroom	59.95	53.95
• Introduction to ClarisWorks—A Tool for Personal Productivity	24.95	22.45
• Math Activities Using LogoWriter—Patterns and Designs	19.95	17.95
• Math Activites Using LogoWriter—Numbers and Operations	16.95	15.25
• Math Activites Using LogoWriter—Investigations	14.95	13.45
Logo for the Macintosh—An Introduction Through Object Logo With the Student Edition of Object Logo	49.95	49.95
The Children's Machine—Rethinking School in the Age of the Computer	22.50	22.50
Object Logo for the Macintosh	195.00	195.00

Receive an additional 20% discount when ordering 10 or more of the same title of ISTE-published products. • Indicates an ISTE-published title.

Name _____ Membership # _____

School/Business _____

Address _____

City _____ State _____ Zip/Postal Code _____

Country _____ Phone _____

Shipping & Handling

$0-$15.99 (subtotal) ..... add $3.50
$16-$45.99 (subtotal) $5.00
$46-$75.99 (subtotal) $6.00
$76-$99.99 (subtotal) $7.00
$100 or more ..... 8% of subtotal

Please do not include *additional site license fees or subscription costs* when computing shipping rates.

GST Registration Number 128828431

LX3

ORDER

Quantity	Title	Member Unit Price	Nonmember Unit Price	Total Price

PAYMENT OPTIONS

☐ **Payment enclosed.** Make checks out to ISTE—
International orders must be prepaid with U.S. funds or credit card.
☐ VISA ☐ MasterCard ☐ Discover Card Expiration Date _____

☐ **Purchase Order enclosed.** Please add $2.50 for order processing—
P.O. not including $2.50 fee will be returned.

☐ **C.O.D.** for U.S. Book orders only. You will pay UPS the total upon delivery by check or cash—
ISTE will add $2.75 order processing.

☐ **Airmail.** International orders for Books & Courseware are sent surface mail—
ISTE will bill you the additional shipping charge for Airmail.

☐ Send me ISTE membership and subscription information.

☐ Send me a free ISTE catalog.

SUBTOTAL	
Subtract 20% on ISTE-published titles if ordering quantities of 10 or more	-
SUBTOTAL	
Shipping and Handling (see box above)	+
Add Additional 5% of SUBTOTAL if shipped to PO Box, AK, HI, or outside U.S.	+
Add 7% of SUBTOTAL for GST if shipped to Canada	+
If billed with purchase order, add $2.50; If COD, add $2.75	+
TOTAL	=

ISTE • 1787 Agate Street, Eugene, OR 97403-1923 • Order Desk 800/336-5191 • Fax 503/346-5890

Looking for Directions?

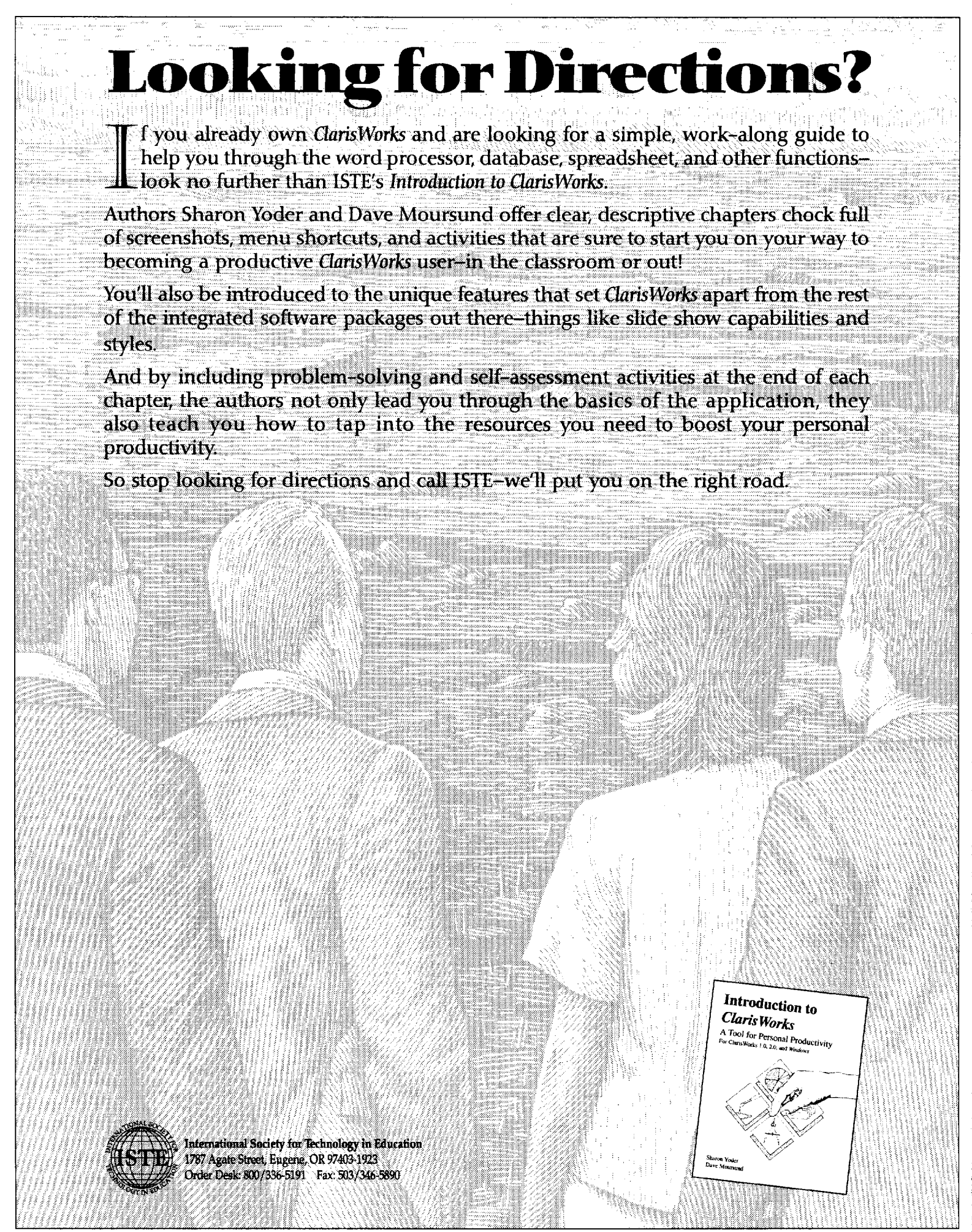
If you already own *ClarisWorks* and are looking for a simple, work-along guide to help you through the word processor, database, spreadsheet, and other functions–look no further than ISTE's *Introduction to ClarisWorks*.

Authors Sharon Yoder and Dave Moursund offer clear, descriptive chapters chock full of screenshots, menu shortcuts, and activities that are sure to start you on your way to becoming a productive *ClarisWorks* user–in the classroom or out!

You'll also be introduced to the unique features that set *ClarisWorks* apart from the rest of the integrated software packages out there–things like slide show capabilities and styles.

And by including problem-solving and self-assessment activities at the end of each chapter, the authors not only lead you through the basics of the application, they also teach you how to tap into the resources you need to boost your personal productivity.

So stop looking for directions and call ISTE–we'll put you on the right road.



International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800/336-5191 Fax: 503/346-5890

"Awesome"
"Way cool."
"Slammin'."
"Totally there."
"Swinging."
"Wicked good."

(And these are just the teachers' comments.)

The fact is, whenever we show our three new educational software products to teachers and curriculum coordinators, they get as excited as kids.

And for good reason.

You see, our line of learning software for Macintosh® computers gives teachers of grades 4-8 a unique way of motivating their students.

For one thing, **MicroWorlds™** products are specifically designed for the classroom. Their flexibility lets students with all different learning styles use what they know to tackle new learning experiences.

What's more, the **MicroWorlds** packages were designed by LCSI, the company known for its award-winning educational products.

Take **MicroWorlds Math Links™** - it doesn't camouflage math as some space game. Instead, it lets you link math to art, science, and social studies. Students don't just study math, they think mathematically, using math to develop projects ranging from kaleidoscopes to Navaho textile patterns.

With **MicroWorlds Language Art™** you'll encourage students to explore words and images. Write text in any shape, color or direction. Add effects such as scrolling text, animation. Projects, including Visual Poetry, Ads, Haiku, help you assist students in developing writing skills.

MicroWorlds Project Builder™ gives you the tools to develop a problem-solving, creative-thinking, learning culture across the curriculum. And features like text, drawing tools, animation, and music give students the tools to create anything from simple ecosystems to dynamic maps.

Plus there's more: Each of these products is offered under LCSI's well-known site/network license - the most flexible policy available to schools today.

So for information or a **free** demo disk, call us today at **1-800-321-5646**.

We think, like, you'll be blown away.

LCSI

MicroWorlds