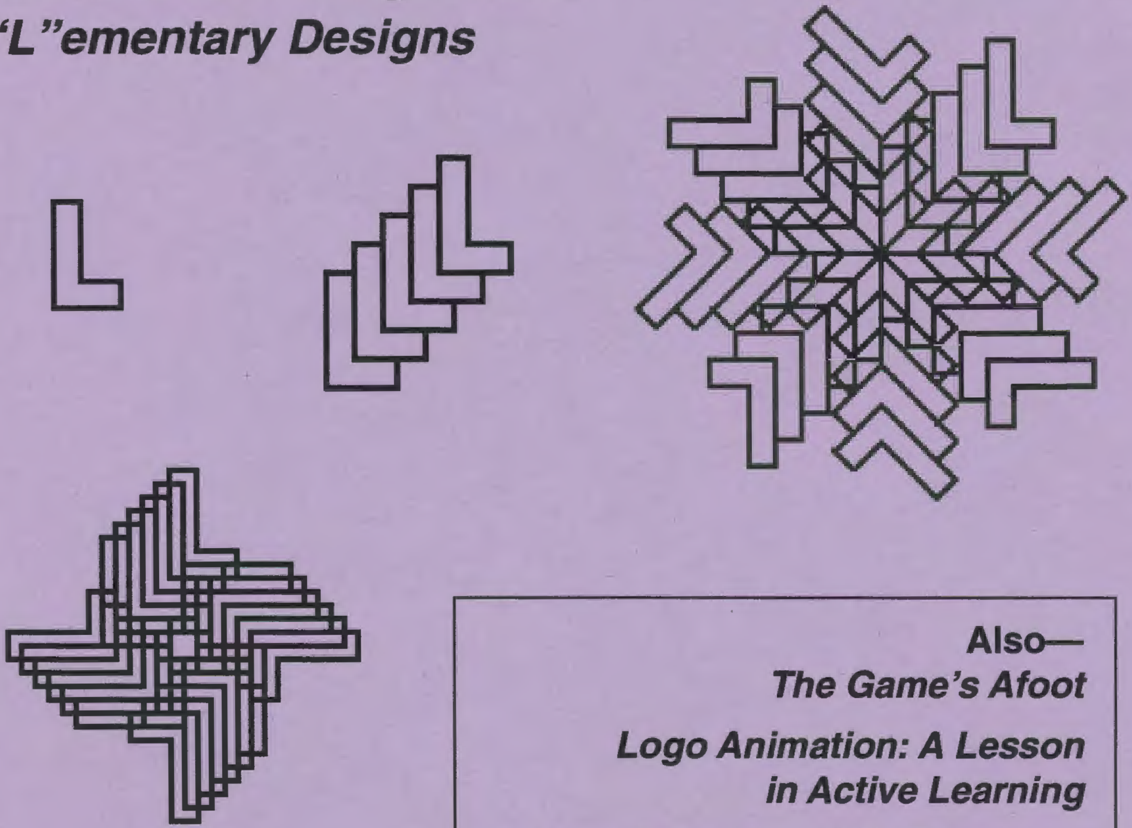
Journal
of the ISTE
Special Interest Group
for Logo-Using
Educators

LOGO EXCHANGE

Fall 1994 Volume 13 Number 1

In this issue: *"L"egant and "L"ementary Designs*



Also—

The Game's Afoot

Logo Animation: A Lesson in Active Learning

Have Some Fun With Recursion

International Society for Technology in Education

LX Volume 13 Number 1

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.

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 14, Number 1	Feb. 1, 1995
Volume 14, Number 2	Apr. 1, 1995
Volume 14, Number 3	July 1, 1995
Volume 14, Number 4	Oct. 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 director of advertising services 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 13 Number 1 Journal of the ISTE Special Interest Group for Logo-Using Educators Fall 1994

Contents

Where *Did* the Summer Go?

by Sharon Yoder

Editor's Note: Due to personal conflicts, staff changes, and production overload, this editorial—written at the end of the summer—is just getting to you. We apologize for the delay, but hope the message here and in the rest of this issue will still be relevant.

As school begins in the fall, most of us find ourselves asking the question: "Where *did* the summer go?" Last June we were full of plans. Perhaps you were going to work in your garden or get outside and exercise every day. Perhaps you were going to take a course or two or learn something new on your own. Maybe you were going to pursue your favorite hobby or start a new one. Most likely you were going to spend some time preparing new materials for school in the fall. And then suddenly it is the end of August and time has run out.

Like many of you, my summer has slipped away with many of my plans unfinished or untouched. Early in the summer my mother became very ill, and I flew across the country to be with her. The days turned to weeks, and in time she died a peaceful and appropriate death. When I returned home with nearly six weeks of the summer gone, I began the process of dealing with the undone tasks and putting aside many of the recreational or personal plans I had made.

... And How Logo Has Changed

As I was thinking about writing this editorial, I began to reflect on how Logo has changed over the years, from a primarily text-based environment to a rather cumbersome triangular turtle to a multiturtled colorful environment to a hypermedia-like environment. And I thought about the conflict between the "Logo purists" and those who enjoy every new form of Logo that comes along—let's call them "Logo futurists."

Somehow, thinking of lost summers reminds me of this long-standing controversy. The purists tell the futurists that their new Logos are too easy—that wonderful problem-solving opportunities are lost in their new Logos. Meanwhile, the futurists tell the purists that they simply don't understand what they are missing—that they are stuck in the past. Both often argue from a negative point of view.

Is the Glass Half Empty or Half Full?

As I reflect on my "lost" summer, I find that it was rich with unexpected blessings. I spent more time in my hometown than I have in many years. I had the opportunity to spend time with friends, some of whom I hadn't talked to for a very long time. I stayed with my father in his apartment in a retirement community and in the process got to know many of his neighbors and gained a whole new "family" of "aunts" and "uncles." I regained a closeness with my father that had gotten lost along the way. I discovered how alike we are in ways I'd never realized before. Even my mother's death had its rich side. I was able to share some very special moments with her and to say a leisurely and quiet goodbye. She had been in pain and unwell for a very long time, so her death was a blessing both for her and for those who loved her.

Take Time to "Smell the Flowers"

When I came home, I found that I had brought with me an ability to slow down, something that my new elderly "family" had taught me. I found that the sewing that didn't get done was still here and would wait until I had time. The book that didn't get written was still there on my desktop waiting for me. The work that must get done is getting done a bit at a time. But most of all, I find that I am valuing every moment of the remaining summer days much more that I ever have. Somehow I have continued to slow down enough to "smell the flowers."

Perhaps it's time for both the Logo futurists and the Logo purists to slow down a bit too and "smell the flowers." There really is no need for the controversy that keeps arising and dividing us. Whatever negative thoughts you may have about another's point of view, there are certainly positive aspects there as well. I think we need to look at the richness that exists in any Logo environment our colleagues choose to use.

When people ask me why I teach desktop publishing the way I do or why I give the kind of hypermedia assignments I do, I generally confuse them by telling them it's because of Logo. Logo really has changed the way I teach. It has changed the way I structure my classroom and my assignments; it has changed the way I present new ideas. Logo has enriched everything I do, and I think that richness has little to do with whether I am a purist or a futurist.

So, take a deep breath and ask yourself what good things happened this summer that you *didn't* expect to happen. What richness are you bringing to your classroom as a result of your experiences? Perhaps the beginning of your school year will be all the better because you can see the gains rather than the losses or failures in your summer.

Making Connections

by Tom Lough

"The toe bone's connected to the ankle bone.
The ankle bone's connected to the shin bone.
The shin bone's connected to the knee bone..."

Hearing students sing this or a similar song brings back memories of childhood experiences. It was fun to explore connections through song. Here are some ideas for connection you might want to suggest to your students as they use Logo.

Connected Procedures

Ask each student to write a Logo procedure to draw some kind of a figure roughly 20 turtle steps wide by about 20 turtle steps long. Each procedure should be named after its author. The last line in each procedure should contain the word "connect." I'll explain in a moment. For example, if Dana decided to draw a star, the procedure would look like this:

```
to dana
repeat 5 [forward 20 right 144]
connect
end
```

Some students might want to write procedures that draw irregular or asymmetrical figures. These procedures would make this activity even more interesting. Here is an example:

```
to terry
forward 20
left 10
back 20
right 20
forward 20
connect
end
```

Collect all the student procedures into a single Logo file for all student groups. (If you have two students named Dana, they will have to figure out how to name each procedure slightly differently.) Let students explore the procedures of the others. Then suggest a class superprocedure that includes the name of several student procedures in turn, for example,

```
to class
dana
terry
juanita
.
.
.
end
```

But what about that **connect** procedure? It's up to you. Here is one to suggest to your students:

```
to connect
forward 40
right 50
end
```

Now they are ready to run the class procedure. After your students discover what the **connect** procedure does, they will probably want to try different **connect** procedures of their own. You might want to suggest the use of random for the turn, such as

```
right random 45
```

Discuss with students how the results of the procedures are connected to each other. What other things can they identify that are connected to each other? Is there anything like the **connect** procedure in their everyday world? Why are connections important?

Connected Functions

If you have students with some previous Logo experience, ask them each to write a Logo procedure that performs one or more mathematical operations on a number and outputs the result. Such procedures are called *operations* or *reporters* in most versions of Logo. Each procedure should be named after its author. Here are a couple of examples you might furnish as starters. Students can make the expression after output as simple or as complex as they want.

```
to kim :number
output :number + 5
end
```

```
    to pat :number
    output 2 * :number + 3
    end
```

In the preceding examples, the variable **number** is included on the same line as the procedure name. This holds a place for a value and also defines **number** as a local variable for the procedure. The **kim** procedure, for example, will output the value of the original number increased by 5.

Encourage students to explore what their procedures can do. In the following example, the **show** command accepts as input and then displays on the screen the output of the **kim** procedure with an input value of 6.

```
    show kim 6
    show pat 5
```

(What do you predict for this one, 13 or 16? This could lead to a very stimulating discussion about mathematical operation precedence.)

Once again, collect all student procedures into a single Logo file for each student group. Give students time to explore the various procedures. Then suggest that they can explore another type of connectivity. Here is an example:

```
    show kim pat 5
```

Was this a surprise? Usually students do not realize that procedures such as this can be linked together in this way. Because the procedures operate as functions, they each accept an input number, operate on it, and then output the result. This result is available as the input to another function.

In the preceding example, pat 5 causes the **pat** procedure to accept the initial value of 5, operate on it, and output its result. Because the **kim** procedure is waiting for an input, it accepts the output of pat, adds 5 to it, and outputs the result, which is displayed by the **show** command.

Initially, students might enjoy deducing what each individual procedure does just from observing the results produced by different input values. Then, they often shift to predicting the results. Invariably, they enjoy setting up, predicting, and then verifying the results of longer chains of procedures, such as the following one:

```
    show robert jill david juanita melva 7
```

What type of personal connectivity does this suggest for people? What things or ideas do we pass on to others that they operate on and pass along to still others? Do any of our behaviors, such as smiles or scowls, get passed along as well? What lessons can we learn from this?

As always, I enjoy hearing from readers who use ideas suggested by this column.

FD 100!

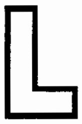
Tom Lough
Founding Editor
Box 394
Simsbury, CT 06070

"L"egant and "L"ementary Designs

by Dorothy Fitch

Welcome to the Beginner's Column! It seems as if I am always playing around with letters in Logo—the shapes that letters make, that is. But the last time I did this in *Logo Exchange* was in 1989, so I don't feel terribly redundant. I figure that if I can be absorbed for several hours by a small idea, it is probably worth sharing. So let's explore!

Here is the letter outline that fascinated me:



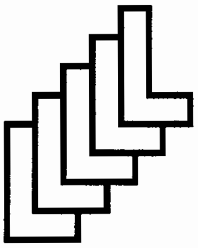
And here is the procedure I used to draw it:

```
TO L
FORWARD 40
RIGHT 90
FORWARD 10
RIGHT 90
FORWARD 30
LEFT 90
FORWARD 15
RIGHT 90
FORWARD 10
RIGHT 90
FORWARD 25
RIGHT 90
END
```

Now that's a straightforward procedure! The turtle starts and ends in the lower left corner. I used a SETPENSIZE 2 2 instruction using *Logo PLUS* for the Macintosh for a thicker line.

Now that we have our L, what can we do? Experiment. Try things. Don't necessarily plan before you begin to type. Just see what happens. To be honest, what I had intended to write about was tessellations, but what I kept coming up with weren't tessellations at all, but other very interesting types of designs. Let's explore a few together.

My first experiments involved nestling Ls against each other, like this:

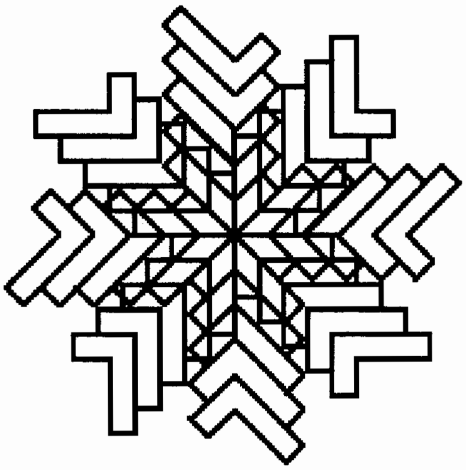


I used a procedure named M to move from one L to the next and called the whole thing a LEG:

```
TO M
PENUP
RIGHT 90
FORWARD 10
LEFT 90
FORWARD 10
PENDOWN
END
```

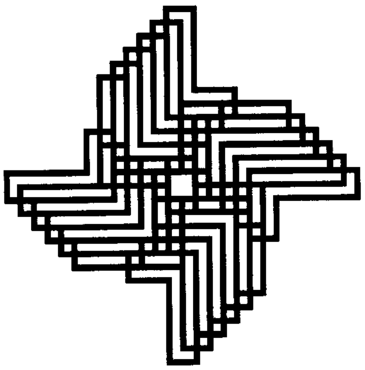
```
TO LEG
REPEAT 5 [L M]
END
```

This experiment quickly led to this star-shaped design with eight legs,

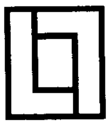


```
REPEAT 8 [LEG PENUP SETXY 0 0 PENDOWN
    RIGHT 45]
```

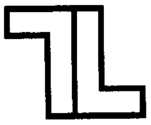
and, with some slight modification to the M procedure, to this design:



Then I tried combining two Ls. I made "innies" where the short part of the Ls faced each other, and "outies," where they pointed away from each other.



"innie"— 2IN



"outie"— 2OUT

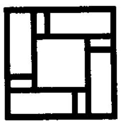
To make a 2IN:

```
REPEAT 2 [L FORWARD 40 RIGHT 90 FORWARD
   35 RIGHT 90]
```

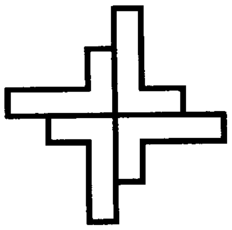
To make a 2OUT:

```
REPEAT 2 [L FORWARD 40 RIGHT 180]
```

And I discovered that four Ls can be combined as "innies" or "outies"! They were even easier to make.



"innie"—4IN



"outie"—4OUT

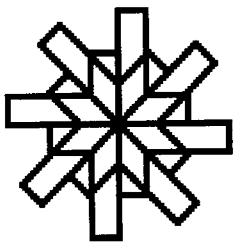
To make a 4IN:

```
REPEAT 4 [L FORWARD 40 RIGHT 90]
```

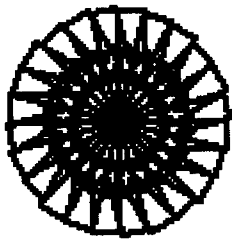
To make a 4OUT:

```
REPEAT 4 [L RIGHT 90]
```

What happens if you make an "outie" with eight Ls?

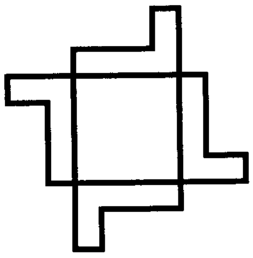


Or 24 Ls?

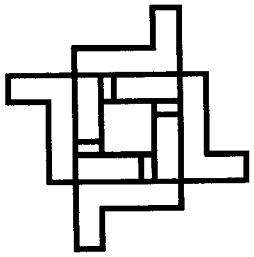


See what your kids encounter as they explore. They can make discoveries about circles and the Total Turtle Trip Theorem as they experiment with Ls.

Here's another way to make an "outie" with four Ls:



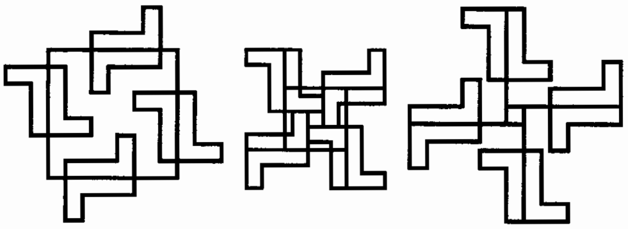
Combining it with the 4IN procedure makes this interesting design:



The following designs are all variations of one instruction, which uses the two-L "outie":

```
REPEAT 4 [2OUT FORWARD 50 RIGHT 90
   FORWARD 25]
```

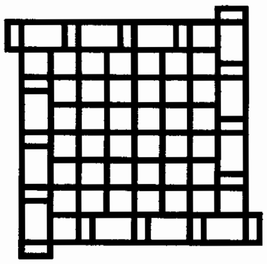
Try changing the input to one or both of the FORWARD commands and see how the design changes. Make the numbers the same. Make them different. Try a negative number!



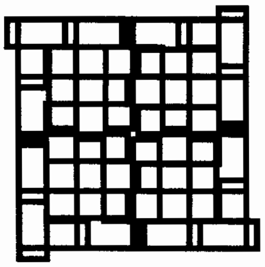
I know that our *LX* editor will take a dim view of my next experiment—I can see her cringing as I write—but here goes. I am generally against nested REPEAT instructions and would never leave one in a finished program, but in this case, using one made my experimentation much easier:

```
    CLEARGRAPHICS
    REPEAT 4 [REPEAT 4 [L PENUP RIGHT 90
        FORWARD 20 LEFT 90 PENDOWN] LEFT 90]
```

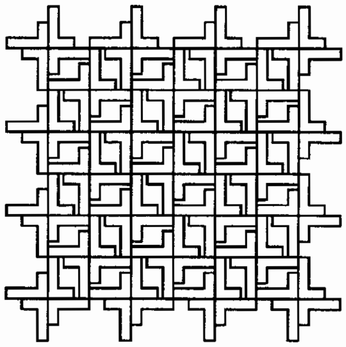
I wanted to play with the input to FORWARD in this instruction, and it saved a lot of typing to nest the REPEATs, even though I was only tampering with one number. To make a long story short, the preceding instruction creates this pleasant design:

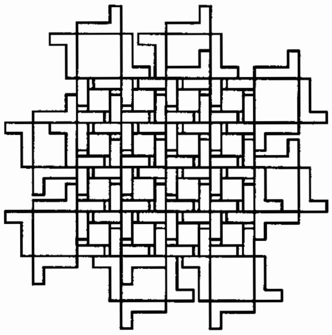


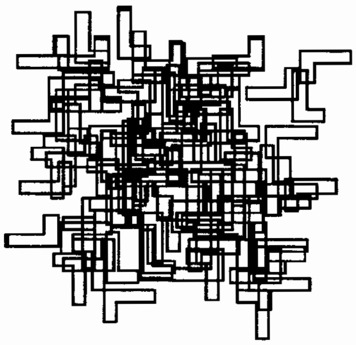
But changing the input to FORWARD just a little creates a very interesting and slightly unsettling effect. I called it BROKEN.WINDOW. The only change is FORWARD 21 instead of FORWARD 20. Try different numbers for different effects.



I could go on and on—which I did!—but I'm running out of space here. I made full-screen L designs: tessellations, things that looked like part of a jigsaw puzzle, and designs that might even hang in the Museum of Modern Art.







As I played with my L, my thoughts bounced from artistic composition to patterns to spatial design to cause and effect, and of course to math and programming. What a lot of thinking one lowly letter inspired!

Happy Logo adventures with Ls—and other equally intriguing letters!

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

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

ISTE DISTRIBUTED

PRESENTS NEW BOOKS AVAILABLE FROM ISTE

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

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.

101 Ideas for Logo

Dorothy M. Fitch, et al.

Logo projects described in *101 Ideas for Logo* include everything from drawing teddy bears to writing adventure games; from planning secret codes to creating physics simulations. There are classic Logo project ideas and new ones that take advantage of more recent additions to Logo, such as shapes and multiple turtles. With projects compatible with almost every version of Logo, *101 Ideas for Logo* offers easy projects for beginners and plenty to challenge expert programmers.

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

For pricing information or to order, see the order form in this publication.

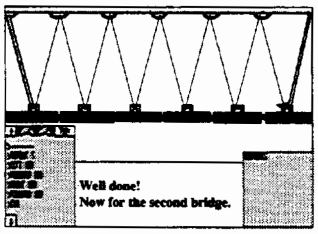
THE CRYSTAL RAIN FOREST

A Mathematical Learning Adventure

The planet Oglo is in trouble. Its rain forests are being destroyed. The king has been poisoned. Only YOU can save them!

The Crystal Rain Forest helps students in grades 3-8 use math to learn about the environment. They hunt for clues in the town, then search for the lifesaving magical crystals deep in the rain forest.

On their quest, they face a series of mathematical puzzles and challenges to solve. They give instructions to robots, guide and rotate shapes to mend bridges, navigate a boat, estimate distances and angles to connect wires, draw shapes to make nets, change box sizes using simple algebra, and so on.



Well done!
Now for the second bridge.

As a result of these carefully sequenced activities, students learn to use the Logo language. *Crystal Logo*, an easy-to-use version, can be run separately from the adventure, and its command names can be modified.

The Crystal Rain Forest, award-winning software from England, is available here as a single user version ($49.95), as a single version for school use with curriculum materials ($59.95), and as a building site license ($250.00).

PC version requires a 286 or better with VGA and a mouse. Mac version requires System 7, color

Terrapin™

TERRAPIN SOFTWARE, INC.
400 RIVERSIDE ST. • PORTLAND, ME 04103
207-878-8200

1-800-972-8200

Palindromes: What Are They and What Can We Do With Them?

by Robert Macdonald

I recall doing the following activity, minus calculator or computer, under my grandmother's tutelage well over 50 years ago. It intrigued me then as it does now. I first introduced it to a fourth-grade class in the fall of 1985, modifying it and learning something new each year. The students learned a great deal, too.

Palindromes

This microworld centers around the investigation of palindromes. A palindrome is a word, a phrase, a sentence, or a number that reads the same forward as it does backward. By extension, we might include as palindromic in character reflective symmetry in the visual arts and melodic inversion or retrograde motion in the aural (Mottershead, 1977).

Thus, as described by Brandreth (1980), "A palindrome is a word, like deed or level or repaper or noon or redder or civic or tenet or kayak or nun, or a phrase or a sentence like 'Madam, I'm Adam,' that reads the same backward as forward."

Numerically, we can visualize 121, 137, 252, a past year (1991), a decade in the future (2002), and 52825 as palindromic. If a number is not initially a palindrome, chances are very good that it may generate a palindrome through inversion (reversion) and addition. For example, take 423. It's fairly evident that 423 is not palindromic. So, let's invert it to 324. That is not a palindrome either. But suppose we add them together:

```
  423
+ 324    reverse and add
  747
```

747 is a palindrome. We have made it a palindrome by the process of reversing the original number and then adding the two numbers together. It is a one-step palindrome—there was one addition—or, as some like to call it, a first-generation palindrome. However, it is not uncommon to go through a number of consecutive reversals (inversions) and additions, for example,

```
   529  is not a palindrome
 + 925  reverse the digits, then add
  1454  is not a palindrome
+ 4541  reverse the digits, then add
  5995
```

5995 is a palindrome. The palindrome was generated by two additions. It is a two-step palindrome, or, if one prefers, a second-generation palindrome.

Multiples

In addition to having some knowledge of palindromes, students should have had some prior instruction on multiples.

Briefly stated, any number is a multiple of any of its factors. For example, 8 is a multiple of 4. It is also a multiple of 2. In addition, other integers may be multiples of 8 (James, 1976):

```
x 1  2  3  4  5  6  7  8
  8 16 24 32 40 48 56 64
```

In the data gathered in the following investigation of palindromes, a knowledge of multiples will increase the appreciation of patterning.

The Use of Calculators

Today there are probably few upper elementary general math texts that do not have a page or two on palindromes created by adding with calculators. The difficulty arrives with those palindromes generated through a number of addition steps. Students easily lose track of counting how many times they have depressed a plus sign or an equal sign. I quickly discovered that a visual check on a computer screen helped produce more reliable data.

For the fun of it, try using calculators to create palindromes for the following numbers: 63 (99, one-step), 86 (1111, three steps), 648 (59895, five steps), 7543 (11011, two steps), and 51716 (247742, two steps).

There is one other activity that might be introduced with calculators at this point—attempting to produce multiples of a number. For example, to produce multiples of 3:

- Clear the calculator.
- Enter 0 + 3 =
- Keep striking the =

You will see 3, 6, 9, 12, 15, 18, and so forth appear in the display window. Following the same patterning, experiment with producing other multiples, especially those of 11. The importance of the multiples of 11 will be revealed shortly.

- First clear the calculator
- Then enter 0 + 11 =
- Keep striking the =

The following multiples of 11 will appear in the display window: 11, 22, 33, 44, 55, 66, 77, 88, 99, 110, 121, and so forth.

You might note that up through 121, all but the number 110 are palindromic.

The Activity

The following microworld activity lends itself to cooperative collaboration. The class is charged with discovering all of the palindromes in the numbers from 0 through 99. A Number Chart is provided (see the sample at end of this article). It is a 10 x 10 matrix of 100 squares, each of which contains a number. The numbers are arranged in the following order:

```
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
```

and so forth, up through 99.

At one time I used a chart of numbers from 1 to 100 arranged in this order:

```
 1  2  3  4  5  6  7  8  9 10
11 12 13 14 15 16 17 18 19 20
```

and so forth, up through 100. However, it proved to be slightly awkward in delineating a very important diagonal of multiples and the relationship of numbers on either side of that diagonal.

A worksheet (see the Data Chart at the end of this article) is also provided to gather data on the palindromes. Students record data for all the numbers from 0 through 99. They record the palindrome generated by the number and the number of steps of addition needed to generate it.

Students should go through the chart and fill in the obvious palindromes. They should be able to complete all 0-step palindromes. A number of them might easily recognize the one-step palindromes.

For example, 55 is already a palindrome; it was generated in 0 steps. Therfore, the student should write 55 as the palindrome and 0 as the number of steps. The number 21 generates 33 as a palindrome in one step. The students should record this information. If solutions are not easily discernible, the calculations can be carried out on a calculator or a computer. (A *LogoWriter* program for doing calculations is provided near the end of this article.)

Using the Data Chart, fill in the Number Chart. In the square under each number, write its palindrome. Lightly color the square to indicate the steps needed for generation. Color the 0-step red, the 1-step blue, the 2-step green, the 3-step orange, the 4-step brown, the 5-step black, and the 6-step yellow. These colors are used for visual orientation.

Prediscussion

There are obvious advantages to using palindromes for a data-gathering microworld. Although the arithmetic task is limited to addition, the possibilities for discovering patterning are unusually great. Higher level thinking skills should be translated into conjecturing, verifying, and stating reliable generalizations.

Obviously, the teacher should prepare a class for the data gathering. I suggest that the class initially be encouraged to recite large numerical palindromes and transform nonpalindromic numbers into palindromes as previously described.

Look at the Number Chart and have the class discuss whether there are more nonpalindromic or palindromic numbers displayed. See if they can transfer this by discerning palindromes from 100-199, from 200-299, and so forth. The following question might be posed: Are there any nonpalindromic numbers that do not generate palindromes? (Suggest the numbers 196 and 8670.)

Somewhere in this discussion, the state of 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 is bound to be brought up. To simplify the results, it is suggested that the student consider them as 0-step palindromes. The *LogoWriter* program near the end of this article identifies them this way, because no reverse addition takes place. Students may wish to consider these single-digit numbers as double digits, the zero holding a tens column open. Thus, they would read 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, making them 1-step palindromes, except for 00. But this may take away from some visual impact on the final colored Number Chart.

Postdiscussion

After students work through the exercise, several points will probably come up for discussion. The class will discover that there are no 5-step palindrones (no black). Obviously, 89 and 98 will cause a stir. They both generate 24-step palindromes. The computer program carries it through the 23rd step. See if the class can solve it from there. The numbers 89 and 98 will remain white. The asterisk at the bottom of the Number Chart indicates a place to write the palindrome. This should also focus attention on the pairing of numbers: 57 to 75, 95 to 59, 79 to 97.

Carefully examine the Number Chart. There is a marked red diagonal going from the top left to the bottom right of the chart: 11, 22, 33, 44, 55, 66, 77, 88, 99. All are multiples of 11. Indeed, all of the 0-step palindromes on the chart are multiples of 11. They continue: 121, 363, 484, 1111, 4884, 44044, 8813200023188.

To the right of the red diagonal, each number inverts itself as it crosses over to the left. Thus, on the right side one finds 19, 36, 69, and 89. On the left side these invert to 91, 63, 96, and 98.

By examining the Number Chart by color, the class will discover that there are 19 zero-step palindromes, 49 one-step, 20 two-step, 4 three-step, 4 four-step, no five-step, 2 six-step, and 2 twenty-four-step palindromes. The grouping of colors and the location of colors are also worth noting.

A Summary Chart at the end of this article provides information that may lead to other generalizations and conclusions. The information has been drawn from the data collected. I originally placed similar charts in some form on large sheets of wrapping paper that were then distributed around the room. I generally avoided blackboards. They can be erased too easily. The material placed on the summarizing charts differed from year to year.

Several years ago one student suggested that the class sum the digits of the numbers used to generate palindromes. He had discovered that the zero-step palindromes moved forward in multiples of 2— 2, 4, 6, 8, 10, 12, 14, 16, 18. The sums of the one-step palindromes moved by ones from 1 through 9. The pattern broke at 10. Two-, three-, four-, six-, and twenty-four-step palindromes followed a ones digit-path from 10 through 17.

Further Explorations

The preceding observations only hint at what a class might recognize. I have always found that the activity, as outlined, occupied as much time as I wished to spend on the topic. However, if students had explored from 0 through 99 one year, it might be tempting at a higher level to explore 100 through 199, or 200 through 299. A group could confront the problem of the number 196. It went to the 24th generation in our computer program before running out of space:

```
  900544455998
+ 899554445009
```

It has been reported that more than 4,000 generations have been tested with no palindrome occuring.

The Computer Program

The program is written for *LogoWriter*. The command for activating the program is **pal**. After that I have programmed the Q key in conjunction with the Control key to facilitate frequent repetition.

```
to pal
clearpage
print [Enter the number you wish to
   turn into a palindrome.]
make "number first readlist
clearpage
palindrome :number
continue
end

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

to palindrome :number
(print char 32 :number)
ifelse :number = invert :number
   [stop][insert "+ insert char 32
   print invert :number]
print [——]
palindrome :number + invert :number
end

to invert :number
ifelse :number = " [output "]
   [output word last :number invert
   butlast :number]
end

to continue
type [If you wish to continue, touch
   the <CONTROL KEY and Q>.]
type char 13
when "Q [Pal]
end
```

The program is user friendly. The user is prompted to input a number. The page clears and the mathematical work is carried out by the computer. If the computer considers a number palindromic, no addition will take place. If generating a palindrome takes much space, the numbers scroll off the screen. To determine the generations (steps) of the palindrome, count plus signs or dashed lines. To count data that has scrolled, use the Up and Down keys to move around the work area. I opted not to put in a counter in the program to provide the number of generations. I wanted students to look at the screen carefully. One may provide too much aid with a computer program.

Those desiring to use a counter might like to look at some programs written by Gary S. Stager (1991). Stager provides other options, such as reporting the record of a certain generation within a group of numbers. Alison Birch (1986) has written a well-formatted program for generating palindromes in the *Logo Project Book.* One might note the manner in which she matches up a line with increasingly larger numbers.

References

Birch, Alison. (1986). *The Logo project book. Exploring words and lists.* Cambridge, MA: Terrapin.

Brandreth, Gyles. (1980). *The joy of lex: How to have fun with 860,341,500 words.* New York: William Morrow and Co.

James, Glenn. (1976). *Mathematics dictionary.* New York: Van Nostrand Reinhold Company.

Mottershead, Lorraine. (1977). *Metamorphosis: A source book of mathematical discovery.* Palo Alto, CA: Dale Seymour Publications.

Stager, Gary. (1991). 1991—The year of the palindromes. *Clime News,* 3(2).

PC **Logo**™
FOR WINDOWS™

It's Here!

A powerful new version of the Logo programming language is now available for Windows.

PC Logo for Windows combines a full-featured Logo with the ease of use of the Windows environment.

Economical multiple-workstation licensing is available.

Exciting robotic connections as well as a full line of Logo materials, books, and curriculum enhance PC Logo for Windows.

Features include:

- **Illustrated, full-color on-line help**
- **Multiple turtles: command separately or together**
- **Turtle color, speed and shape control**
- **Supports Windows installed fonts and printers**
- **Compatible with other Windows programs**

HARVARD
ASSOCIATES, INC.

For more information or to order, call:
800-774-LOGO

10 Holworthy St. • Cambridge, MA 02138 • Fax (617) 492-4610
CompuServe 70312,243 • Internet 70312.243@compuserve.com

teacher certified
NEA™

Data Chart

Name ______________________________ Date: __________

No	Palindrome	Steps	No	Palindrome	Steps	No	Palindrome	Steps	No	Palindrome	Steps
0			25			50			75		
1			26			51			76		
2			27			52			77		
3			28			53			78		
4			29			54			79		
5			30			55			80		
6			31			56			81		
7			32			57			82		
8			33			58			83		
9			34			59			84		
10			35			60			85		
11			36			61			86		
12			37			62			87		
13			38			63			88		
14			39			64			89		
15			40			65			90		
16			41			66			91		
17			42			67			92		
18			43			68			93		
19			44			69			94		
20			45			70			95		
21			46			71			96		
22			47			72			97		
23			48			73			98		
24			49			74			99		

Number Chart

Name ______________________________ Date: __________

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89*
90	91	92	93	94	95	96	97	98*	99

Color: * ______________________________

Red = 0-Step Palindrome	Brown = 4-Step Palindrome
Blue = 1-Step Palindrome	Black = 5-Step Palindrome
Green = 2-Step Palindrome	Yellow = 6-Step Palindrome
Orange = 3-Step Palindrome	

Summary Chart

(Single-digit numbers are excluded.)

Palindrome Generated	Number Used to Generate Palindrome	Steps	Sum of Digits of Palindrome	Number of Times Palindrome Appears
	11	0	2	
11	10	1	1	2
	22	0	4	
22	20	1	2	2
	33	0	6	
33	12,21,30	1	3	4
	44	0	8	
44	13,31,40	1	4	4
	55	0	10	
55	14,23,32,41,50	1	5	6
	66	0	12	
66	15,24,42,51,60	1	6	6
	77	0	14	
77	16,25,34,43,52,61,70	1	7	8
	88	0	16	
88	17,26,35,53,62,71,80	1	8	8
	99	0	18	
99	18,27,36,45,54,63,72,81,90	1	9	10
	29,38,47,56,65,74,83,92	1	11	
121	19,28,37,46,64,73,82,91	2	10	16
363	39,48,57,75,84,93	2	12	6
484	49,58,67,76,85,94	2	13	6
1111	59,68,86,95	3	14	4
4884	69,78,87,96	4	15	4
44044	79,97	6	16	2
8,813,200,023, 188	89,98	24	17	2

"The Game's Afoot!" or "It's Still a Mystery to Me"

by Christine A. Johanek

"I love to lose myself in a mystery, to pursue my reason to an 'O altitudo!'"

—Sir Thomas Brown, *Religio Medici*

One of the most enjoyable units in a secondary English class is that of "Detective Fiction" or "Classic Mysteries." Through reading and discussion of various novels and short stories, students are given the opportunity to "solve" crimes alongside some of the greatest sleuths of all time: Holmes, Poirot, Miss Marple, Ellery Queen, Dupin. Although some students are able to solve the crime *with* the sleuth, most are left scratching their heads and asking, "How did he know *that*?" or "How did she know something so small was a clue?"

"It has long been an axiom of mine that little things are infinitely most important."

—Sherlock Holmes, *A Case of Identity*

In studying the characteristics of detective fiction, all students learn the requirements of a classic mystery:

1. A crime must have been committed (no fair if the victim dies of natural causes).
2. The crime and solution must not involve the supernatural (no ghosts allowed).
3. There must be a motive (a senseless, random crime does not a classic mystery make).
4. The criminal must be a character in the story (a new character/criminal cannot be introduced when the crime is solved).
5. All clues for solving the mystery must be presented to the reader as well as to the fictional detective (the detective cannot have hidden information).

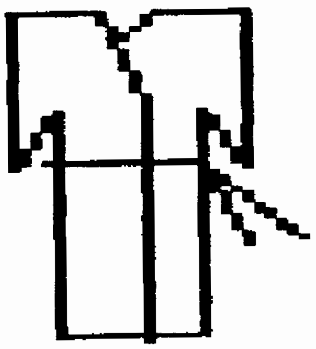
The first four characteristics present little problem for students; these are simply the "rules of the game." It is the fifth characteristic that can present a challenge to readers. When is a clue *not* a clue? (When it is a red herring.) When do seemingly inconsequential details point to the murder? In solving mysteries, it's the application of the powers of observation and deduction to many bits of information that separates the "men" from the "boys"—or rather, the "Holmeses" from the "Watsons."

A simple Logo program can help students sharpen their powers of observation and deduction. The procedures for VILLAGE.MURDERS were developed to augment the reading of Agatha Christie's "Village Murders," which features the detective Miss Marple. The same processes could be applied, however, to any number of other classic mystery stories and novels. The plan of the Logo-Mystery lesson is as follows:

1. Students read "Village Murders" by Agatha Christie up to the point where "Miss Pollit promised to see what she could do." At this point in the story, all essential clues to solving the crime have been revealed. The identity of the murderer and the denouement—or "unraveling"—is all that remains.
2. The students adjourn to the computer—individually or in groups, during class or during free time. After typing the command VILLAGE.MURDERS, the student is presented with the following directions:

```
LET'S SEE HOW WELL YOUR POWERS OF
OBSERVATION WERE USED IN "VILLAGE
MURDERS." MISS MARPLE NEEDS HELP
TO SOLVE THE CRIME. THERE ARE
ELEVEN CLUES TO HELP DEDUCE THE
MURDERER. HOW MANY CAN BE FOUND?
TYPE IN ONE CLUE.
```

3. The student supplies possible clues (commands) which, if correct, will draw the clue as a visual reminder. For example, the student may enter the clue KIMONO.

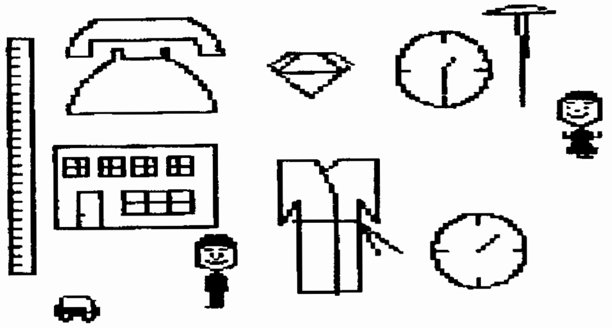


If the clue is incorrect, the Logo mantra "I don't know how to ..." appears. And the student must try again. For convenience, the procedures have been given several similar command names, any of which the student may use. A complete list of procedures for drawing the clues in this story is contained on a listing of procedures available on the disk mentioned at the end of this article.

4. After the students have found all 11 clues—or have given up—they can QUIT and enter the number of clues found for an instant evaluation. The following message appears if a student finds nine clues:

```
HOW MANY TOTAL CLUES DID YOU
FIND? TYPE THE NUMBER WORD (NOT
THE NUMERAL).
NINE
NINE CLUES! POTENTIAL IS THERE TO
BE A MASTER SLEUTH. POWERS OF
OBSERVATION ARE MAXIMUM.
NOW CHECK WITH THE TEACHER FOR
FURTHER INSTRUCTIONS.
```

5. Each evaluation statement contains the direction "Now check with the teacher for further directions." Perhaps the teacher will ask for the student to use PRINTSCREEN to provide a hard copy for the teacher's evaluation needs; perhaps the student will be given some "nudges" and directed to reread key passages. All this is up to the individual teacher.
6. If directed by the teacher, students can self-check their work by typing the command CLUES.



The graphic representation of the clues appears, but not the commands. The student can then continue trying "clues" (commands) to reproduce the graphic.

7. The students return to the classroom setting for closure. They compare the clues they have found, speculate on the resolution of the mystery, and read how Christie's Miss Marple solves the murder. Related activities are limited only by the time available for mystery activities. Here are some possibilities:
   1. After identifying the clues, the students write their own resolutions to the mystery.
   2. The student reads another mystery independently and develops a similar Logo program to be used by others in the class.
   3. The students write original mystery stories. To check for "reasonable" and "workable" clues, each student writes a Logo program identifying the clues in the story.
   4. Students evaluate each other's stories by using the program constructed in Item 3. Are the clues logical and easily identified?

As Sherlock Holmes remarked in *The Bascombe Valley Mystery*, "Good mysteries seem to be cases ... which are so extremely difficult." To which Watson replied, "That sounds a little paradoxical. But it is profoundly true. Singularity is almost invariably a clue. The more featureless and commonplace a crime is, the more difficult it is to bring home."

While the program described here and others like it cannot teach students to solve all mysteries—after all, that's what fictional detectives are for—it can help them identify important clues or information presented in literature. By honing their skills of observation and deduction with this and similar programs, students can develop a "sixth sense" for mystery.

If you would like a copy of the Village Mystery program, please send a self-addressed stamped disk mailer and a 5.25" disk for the Apple IIe to Christine Johanek at the address given below.

Christine Johanek teaches sophomore and senior English at Daniel J. Gross High School in Omaha, Nebraska. She also coaches varsity cheerleading. In her spare time she enjoys traveling, does cross stitching, and sings in her church choir.

4921 Vinton Street
Omaha, NE 68106

Logo Animation: A Lesson in Active Learning

by Kathleen H. McClaskey

In the fall of 1989, I was searching for a Logo project that my eighth-grade students would find both challenging and engaging. Their previous experience in Logo was a four-week introductory lesson to Logo programming in the seventh grade, where they learned a number of Logo primitives and the concept of a superprocedure. I wanted to create a lesson that would build on the Logo knowledge they acquired in the seventh grade and expand their problem-solving abilities in the eighth grade.

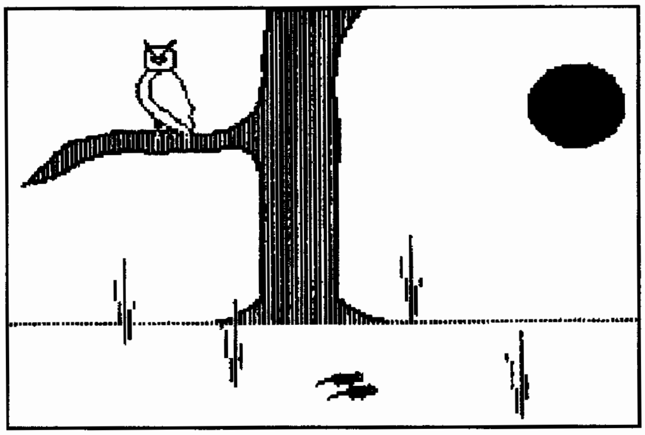
The summer prior to the 1989 school year I decided to purchase *Logo PLUS,* which I had seen demonstrated. I had found its additional commands and turtle-shape capabilities quite appealing. I also acquired a reference book, *Introduction to Programming in Logo Using Logo PLUS,* by Sharon Burrowes Yoder (1989). I came across a section in this book called Animation, which immediately excited me as a potential project. I began my venture into designing a lesson that included a number of new commands found in *Logo PLUS,* especially those that my students needed to understand to design and program their own animation.

Back in 1989, when I introduced this lesson, students were turned on by their ability to animate turtles as well as change the turtle's shape, color, and size. The new commands I introduced initiated a self-discovery that sparked the imagination of each of my students. They in turn became active participants in their learning. After two weeks of introducing *Logo PLUS* (which included a review of Logo commands and superprocedures they had learned in the seventh grade), my students were ready to embark on one of the most engaging projects they would ever experience in a classroom—one they could not wait to get started on. After presenting the project criteria, we discussed the use of a superprocedure and came up with a basic program design that everyone could use for their animation project.

Animation Project

Week 1

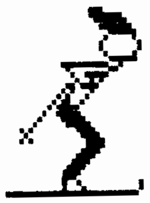
The students first sketched the animation on paper. Then I consulted with each of them on the approach they would take to program it. The first component the students worked on was the landscape or background in which their turtle(s) would be animated. The following landscape was created using only one turtle:

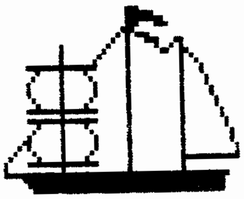


Most students designed their landscapes with anywhere from three to six items. I stressed the importance of creating subprocedures so that the program would be easier to debug. From that time on, I became a consultant when there were questions on how to program. At that time, I would sit one-on-one with a student and discuss what he or she would like to do, which often provided a programming approach that would work. I also stressed the importance of asking other students to assist in programming problems and to listen closely to suggestions. Programming a landscape took students approximately five to seven class periods. As the students' confidence level in programming increased, so did the number of landscapes and the number of turtles they wanted to animate.

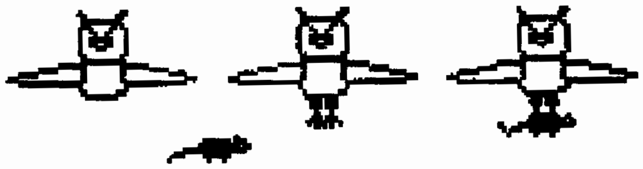
Week 2

After the landscape was finished, it was time to actually animate the turtle in the scene. When I first introduced this lesson, students did not venture into creating their own turtles, but by the middle of the school year the majority of the students wanted to create their own turtles and often created many of them. Students became so involved in making their own turtles that I can hardly remember when an original turtle was not used in an animation.





The second week involved designing the turtles they wanted to use in the animation, locating them in their landscape, and creating turtle movements that would give life to their animations. Many students created anywhere from three to six turtles and would either animate them or stamp them to make them part of the landscape.



Many students became so motivated and involved with their projects that they spent numerous hours in study periods and after school working on them. In fact, I had one student who had the software at home and programmed several minutes of sound effects for his animation.

Week 3

The final week of the project provided the opportunity for students to put finishing touches on their landscapes or animated turtles and to demonstrate their projects for me. I spent time with each student, discussing the process they had gone through to complete their animations. The students were so thrilled with their unique animations that they readily showed them to their classmates. I also taped their animations on video to share with future classes and to demonstrate in teacher workshops.

Assessment of Student Outcomes

I believe that assessing the student outcomes of a project is the most important part of any lesson. Creating animation with *Logo PLUS* is genuinely engaging because every student is empowered to create his or her own animation and then program it to work according to the original design. This lesson in problem solving is what I call an "active learning/adventurous teaching" experience, where all students, from the gifted to the at-risk, find success because they take ownership of their learning. Practicing the well-known Logo programming motto "Ask Three Before Me," students openly collaborated, discussed their programming problems with other students, and discovered the value of listening to other students' ideas on the various approaches they could take. It became apparent to me that the dynamics of peer collaboration, the one-on-one coaching I did with each student, and the individual PEP (process, effort, and programming techniques) assessment contributed to an outcome of self-esteem and sense of achievement. In fact, I was able to document that the at-risk students involved in this project had many more successes in other academic classes with this renewed self-confidence.

I also attribute the success of this lesson to the fact that students have a natural curiosity about how objects move on a computer screen. To their amazement, they were able to create their own animation. Most importantly, they proved to themselves that hard work can be very rewarding.

Anyone interested in knowing more about this fantastic project may contact me by voice, mail, or e-mail. I would like to hear from other "adventurous" teachers.

Reference

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

Kathleen H. McClaskey taught at the elementary and junior high level where she developed curriculum using Logo and *Logo PLUS*. She has been an educational technology consultant in New Hampshire, where she has given workshops in animation with Logo and in distance learning. For the past two years, her focus has been on distance learning, working with an organization called BEADL—Business and Educational Alliance for Distance Learning—which is designed to provide service, information, and support in distance learning technologies to schools and educational institutions throughout the Northeast. Currently she is a technology coordinator for a middle school in Massachusetts.

4 Arrow Lane
Amherst, NH 03031
603/424-7589
Internet: K_McClaskey@aol.com

Turtle and Coordinate Geometry in an Elementary Environment

by Robert Macdonald

Students interested in Logo can easily be introduced to two ways of constructing figures on a plane: turtle geometry and coordinate geometry. Turtle geometry has been described as "the geometry of forward and right" (Abelson & di Sessa, 1980, p. 11). Seymour Papert described turtle geometry as utilizing "body-centered" or "body-syntonic" commands. Students experimenting with turtle graphics are conscious of its power. Most adults are familiar with the use of Cartesian coordinates in geometry. Indeed, the combining of aspects of algebra and geometry in these coordinates is one of the great accomplishments of the great French philosopher and scientist René Descartes (1596–1650). (Cartesian is derived from the Latinization of his familial name—Renatus Cartesius.) His great interest in applying mathematical methodology to all aspects of human knowledge was the basis of his philosophy. He cast aside the system of the scholastics and substituted doubt. This is probably best expressed in his most famous quote: Cogito, ergo sum (I think, therefore I am). He died under mysterious circumstances while at the court of the enigmatic Queen Christina of Sweden.

Comparing Turtle and Coordinate Geometry

In many respects turtle geometry, with its emphasis on body position, is more useful to the younger learner because the position of the turtle is not inordinately fixed but is highly portable. Cartesian coordinates have a disadvantage in being inflexible. On occasion this may be a great advantage, but not necessarily to the younger learner.

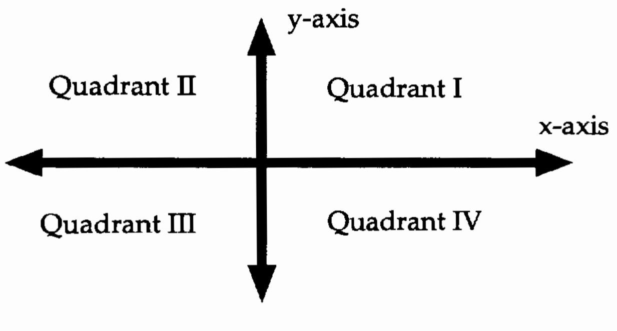
In turtle geometry the turtle's movement commands are relative to the turtle's current heading and position. Graphics utilizing Cartesian coordinates assume an absolute reference frame. We may use "body-centered commands" to position a turtle in turtle geometry and then use a command to discover its position in Cartesian coordinates. Throughout this article we shall investigate this feature in greater depth.

The Coordinate System

Cartesian coordinates work on a two–dimensional plane. To locate a point on that plane we need two numbers. One number, the first, informs us how far a point is to the right or to the left; the second number indicates how high or low it is. Students readily see a relationship to the latitude and longitude they have previously studied in geography.

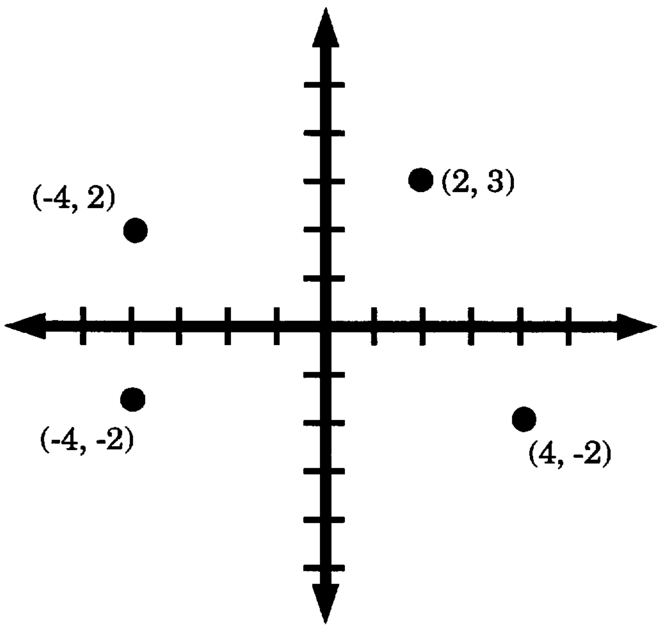
To locate the necessary two coordinates, we need some fixed lines of reference: an x–axis and a y–axis. These axes will meet at a point called the origin, which is represented by the coordinate pair (0, 0). It is from this point of origin that we define our points. The first number (representing a position on the x–axis) will define the horizontal distance from the origin to the desired point. The second number (representing the y–axis) will define the vertical distance from the origin to the desired point. A positive x-coordinate indicates that the point will be located to the right of the origin. A negative x-coordinate indicates that the point is located to the left of the origin. In like fashion, a positive y-coordinate indicates that the defined point is above the origin, while a negative y-coordinate places it below the origin.

The technicalities finished, let's become very practical. By the time children arrive in fourth grade almost all have worked with paired numbers in the first quadrant. The two axes may be presented with an analogy to geography: the x–axis represents the equator; the y-axis is represented by the prime meridian. Let's draw the axes and label the four quadrants that are formed.



The point at which the axes meet is the origin. In Logo it is the home position of the turtle [0, 0].

Let's locate some points:



If you look closely at these quadrants, the clarity of Cartesian coordinates is evident. The point where the axes cross is called the origin. From this point all other points are determined. The first quadrant is always represented by positive numbers, (2, 3) in our example. The second quadrant is represented by a negative number followed by a positive number: (-4, 2). The third quadrant displays two negative numbers: (-4, -2). The fourth quadrant displays a positive number followed by a negative number: (4, -2).

I have always found it useful to provide students with a laminated grid on which they might locate points with a water–based pen. Provide a small, moist sponge to remove the points and the lines connecting them when necessary. An example of such a grid is provided at the conclusion of this article. But before having the students build figures using coordinate geometry, let's make a study of some important computer commands that may be of great use to them.

Useful Logo Commands in Coordinate Geometry

Experimenting with the following primitives will prove beneficial to students in the work to be presented shortly. If you are working with *LogoWriter* on the Macintosh, drag the turtle to some position within Quadrant II. (If you are using an Apple IIe or IIGS use the Turtle Move mode.) To discover the position of the turtle, type

```
show pos
```

in the Command Center. In our example, [-83 35] appears. You can then use this pair of numbers in a **setpos** command, for example,

```
setpos [-83 35]
```

If you wish to discover only the position of the x–coordinate, type

```
show xcor
```

in the Command Center. The number -83 appears. If you need to know only the y–coordinate, type

```
show ycor
```

in the Command Center. The number 35 will appear.

You may also set each of the coordinates independently. For example, to set an x–coordinate, type

```
setx 40
```

in the Command Center. To verify the change, type

```
show pos
```

The coordinate pair

```
[40 35]
```

appears. Now try to change only the y–coordinate:

```
sety -30
```

Verify the results with

```
show pos
```

The coordinate pair

```
[40 -30]
```

appears. To discover the size of the plane on the computer screen, experiment with the preceding primitives.

Programming With Coordinates

To ascertain the progress students are making in working in all four quadrants, I like to have them create a program much like the following one. First, the axes must be drawn and some figure must be programmed in each of the four quadrants. The simple program is written in *LogoWriter* for the Macintosh. **Program** runs the program.

```
to program
prepare
axes
quadrant1
quadrant2
quadrant3
quadrant4
end
```

```
to prepare
if not front? [flip]
rg
ct
cc
end
```

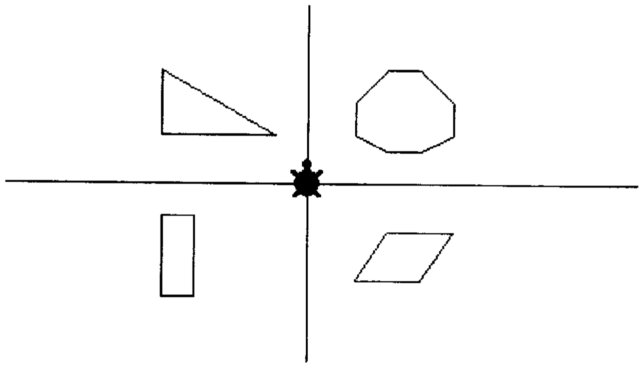
```
to axes
forward 110        (forward 85 on IIe, IIGS)
back 220           (back 170 on IIe, IIGS)
home
left 90
forward 250        (forward 140 on IIe, IIGS)
back 500           (back 280 on IIe, IIGS)
home
end
```

```
to quadrant1       (This subprocedure will
                   make an irregular octagon.)
pu
setpos [30 30]
pd
setpos [50 20]
setpos [70 20]
setpos [90 30]
setpos [90 50]
setpos [70 70]
setpos [50 70]
setpos [30 50]
setpos [30 30]
pu
home
end
```

```
to quadrant2       (This subprocedure draws a
                   triangle.)
pu
setpos [-20 30]
pd
setpos [-90 70]
setpos [-90 30]
setpos [-20 30]
pu
home
end
```

```
to quadrant3       (This subprocedure draws
                   an oblong rectangle.)
pu
setpos [-90 -20]
pd
setpos [-90 -70]
setpos [-70 -70]
setpos [-70 -20]
setpos [-90 -20]
pu
home
end
```

```
to quadrant4       (This subprocedure draws a
                   parallelogram.)
pu
setpos [50 -30]
pd
setpos [90 -30]
setpos [70 -60]
setpos [30 -60]
setpos [50 -30]
pu
home
end
```



Have students use a laminated grid to create the figures for their subprocedures. The program is not complex. Analyze each of the subprocedures to see what each accomplishes. If you do not wish to use turtle graphics to set up the axes, do use coordinates to set up the outer limits of the grid. This task is not very difficult.

Further Work in Each Quadrant

I like to provide exercises such as the following to point out the inflexibility of a coordinate system. We will build a sailboat in each of the quadrants and one additional sailboat that uses all the quadrants. Have students construct a similar sailboat by using turtle graphics. The program using turtle graphics should be portable to each of the quadrants and even use points in all four quadrants.

```
to sailboat1
```
This procedure locates the sailboat in the first quadrant.

```
rg
axes
ht
pu
setpos [80 50]
pd
setpos [100 30]
setpos [10 30]
setpos [90 110]
setpos [140 30]
setpos [100 20]
setpos [20 20]
setpos [40 0]
setpos [110 0]
setpos [130 20]
setpos [100 20]
pu
home
end
```

```
to sailboat2
```
This procedure locates the sailboat in the second quadrant.

```
rg
axes
ht
pu
setpos [-70 50]
pd
setpos [-50 30]
setpos [-140 30]
setpos [-60 110]
setpos [-10 30]
setpos [-50 20]
setpos [-130 20]
setpos [-110 0]
setpos [-40 0]
setpos [-20 20]
setpos [-50 20]
pu
home
end
```

```
to sailboat3
```
This procedure locates the sailboat in the third quadrant.

```
rg
axes
ht
pu
setpos [-70 -60]
pd
setpos [-50 -80]
setpos [-140 -80]
setpos [-60 0]
setpos [-10 -80]
setpos [-50 -90]
setpos [-120 -90]
setpos [-100 -110]
setpos [-40 -110]
setpos [-20 -90]
setpos [-50 -90]
pu
home
end
```

```
to sailboat4
```
This procedure locates the sailboat in the fourth quadrant.

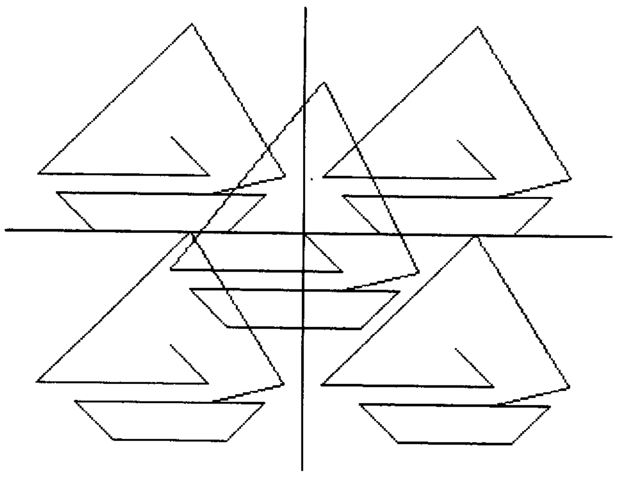
```
rg
axes
ht
pu
setpos [80 -60]
pd
setpos [100 -80]
setpos [10 -80]
setpos [90 0]
setpos [140 -80]
setpos [100 -90]
setpos [30 -90]
setpos [50 -110]
setpos [110 -110]
setpos [130 -90]
setpos [100 -90]
pu
home
end
```

```
to sailboat5
```
This procedure uses points in all four quadrants.

```
rg
axes
ht
setpos [20 -20]
setpos [-70 -20]
setpos [10 80]
setpos [60 -20]
setpos [20 -30]
setpos [-60 -30]
setpos [-40 -50]
setpos [30 -50]
setpos [50 -30]
setpos [20 -30]
pu
home
end
```

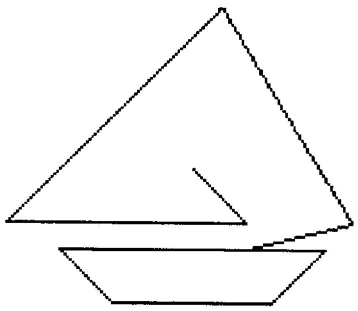
The following program uses the preceding procedures to place a sailboat in each quadrant and then uses points in all of the quadrants to construct a sailboat.

```
to allboats
sailboat1
wait 60
sailboat2
wait 60
sailboat3
wait 60
sailboat4
wait 60
sailboat5
end
```



The following is an example of how to construct a sailboat by using turtle graphics. The program **boat** will accomplish the task.

```
to boat
right 145
forward 20
right 125
forward 80
right 130
forward 90
right 100
forward 100
right 120
forward 30
right 10
forward 90
left 145
forward 30
left 35
forward 70
left 45
forward 25
left 135
forward 20
ht
seth 0
end
```



The preceding program can be used to distribute boats in each of the four quadrants. We merely locate a starting point with Cartesian coordinates. The program demonstrates the flexibility of turtle geometry over the coordinate geometry we used in the **sailboats** program. The command for the following program is **boats**:

```
to boats
rg
axes
pu
setpos [114 52]
pd
boat
pu
setpos [-139 52]
pd
boat
pu
setpos [-134 -60]
pd
boat
pu
setpos [134 -60]
pd
boat
end
```

A Useful Little Program

The following program is useful when experimenting with Cartesian coordinates:

```
to p :x :y
setpos (list :x :y)
end
```

Try the coordinates in the previously listed **quadrant4** procedure. Type the command **p**—a shortened form of point—and input the coordinates in each **setpos** command. This is an easy way of checking what has been drawn on a grid, for example,

```
pu
p 50 -30
pd
p 90 -30
p 70 -60
p 30 -60
p 50 -30
ht
```

When work is done in the Command Center, we can verify our work. Try some of the other figures drawn through the application of coordinates.

Combining Turtle and Coordinate Geometry

Rarely is anything pure. In the preceding program we resorted to both turtle and coordinate geometry to ease the difficulty of programming. Let's demonstrate with another example. A simple program producing random squares on the computer screen is a good example. This program was written using *LogoWriter* for the Macintosh. The command is **random.squares**.

```
to random.squares
rg
repeat random 100 + 1 [pu setpos
   random.points pd square random
   100 + 1]
ht
end

to random.points
output list (random 121) - 80
   (random 121) - 100
end

to square :size
repeat 4 [forward :size rt 90]
end
```

In this program, the subprocedure for producing the square is a typical example of applying turtle graphics. The coordinate points for locating each square are randomly selected in the subprocedure **random.points**. What we need here is a way to produce random locations in both a positive and negative range. I decided to use a random 121 for both my x-coordinates and y-coordinates. This gives a range of selection from 0 through 120. I decided to subtract 80 from random 121 on the x–coordinate and 100 on the y–coordinate. This produces some random negative points around the four quadrants.

Play around with the numbers if you want to produce a wider display. My selection helps keep the squares drawn near the center. I've been rather pleased with some of the outputs. In the superprocedure **random.squares** you will discover why I wrote **random.points** as a procedure that outputs a list comprising coordinate points. It was the only way I could get the **setpos** command to accept the randomly generated coordinates. You might also like to play around with this aspect of the program. This is what makes programming so much fun. How do you accomplish what you want? Notice also the number of **random** reporters used in **random.squares,** for example,

```
repeat random 100 + 1 [pu setpos
   random.points pd square random 100
   + 1]
```

The **random** inputs for both repeat and square will generate a random selection from 1 through 100. I did not want to take the chance that an input of 0 would be generated. If you have a color screen you might like to put a **setc** command with a random input. The more you experiment, the more you learn.

Playing Around With Follow the Dots

Necessity is the mother of invention. Several years ago a group of my fourth graders volunteered to help out a first-grade class. The group decided it would like to prepare some sheets of "follow the dots" for their first graders, using some of the material the fourth grade had worked out with Cartesian coordinates. To carry out the task I decided to write a short program that would graph the points by using coordinates. The result was the following **graph** program:

```
to graph
pu
show [What point do you want to
   graph?]
setpos readlistcc
pd
dot          <—You may wish to substitute the
             procedure cross.
show [Label your point.]
label readchar
show [Another point? y or n]
if readchar = "y [graph]
end
```

Some of my students preferred this procedure:

```
to cross
forward 4
back 2
right 90
forward 2
back 4
forward 2
left 90
end
```

You will have to create a small circle on the shapes page if you want to use the following procedure rather than the **cross** procedure:

```
to dot
setsh 1
stamp
end
```

When the program is run, you will be asked to enter the points to be graphed. Simply enter the two numbers separated by a space. No square brackets are required. Another request asks for the label you wish to give that point. Input a number or a letter and remember to sequence them logically. Then you will be asked if you wish to graph another point. Continue until you have finished your own edition of "follow the dots." Students will enjoy applying their newly gained knowledge in this manner.

References

Abelson, Harold, & di Sessa, Andrea. (1980). *Turtle geometry. The computer as a medium for exploring mathematics.* Cambridge, MA: MIT Press.

Yoder, Sharon. (1991). *Introduction to programming in Logo using* LogoWriter. (2nd ed.) Eugene, OR: International Society for Technology in Education.

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

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

A First Course began as a curriculum for our own classes. Today it is used around the world in hundreds of school districts.

The comment we hear most often is, "These materials were obviously written by classroom teachers."

Secondary as well as elementary teachers have found these materials to be a valuable resource.

This fall will be our eighth year of teaching Logo. We hope all who teach Logo enjoy it as much as we do.

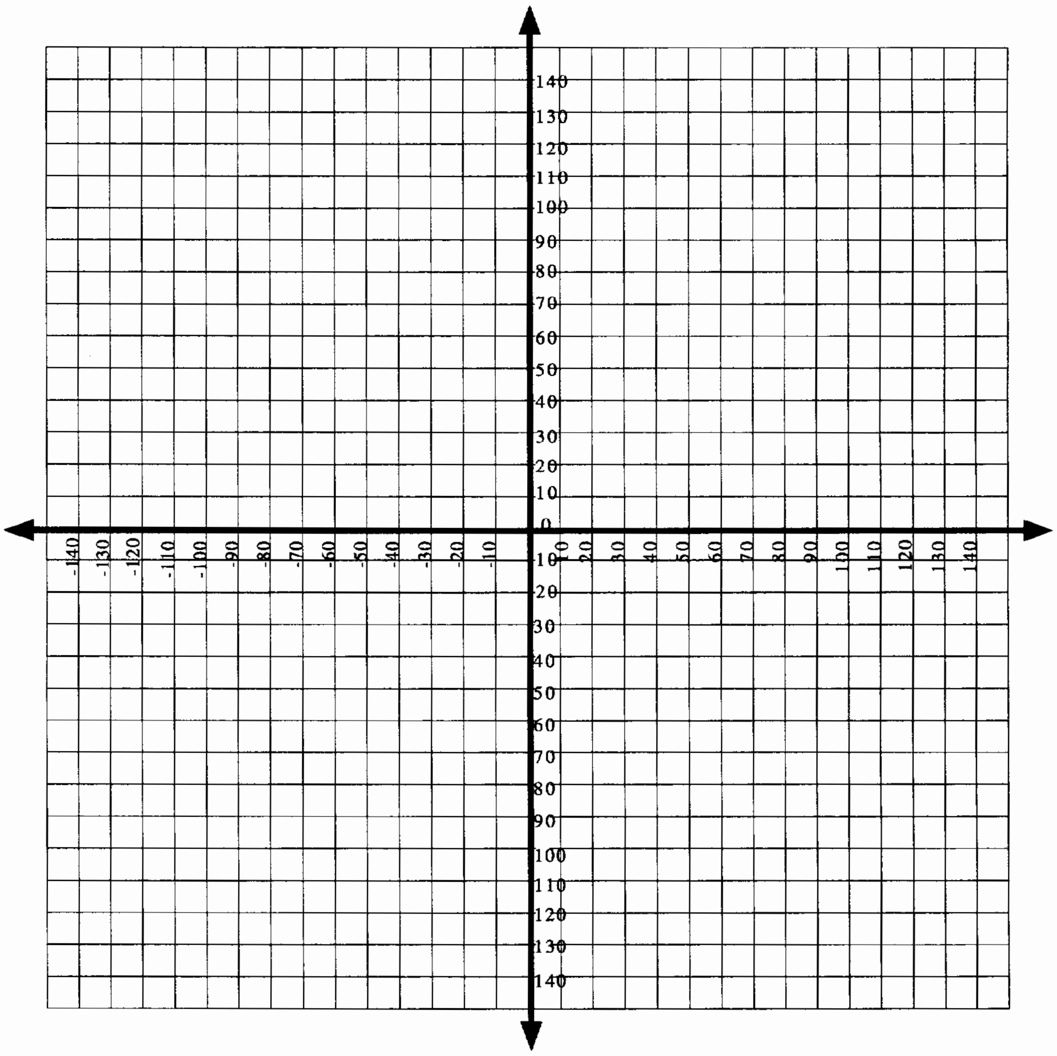
A First Course in Programming is a directed learning environment in structured programming. Its 450 pages emphasize problem solving strategies, critical thinking skills and solid principles of computer science.

This is a complete curriculum for a semester course in programming. It includes student activity sheets, teacher lesson plans, tests, quizzes, assignments, and sample solutions for all student assignments (hard and soft copy!). Only $295 for a building site license. For information or orders contact:

Logo Curriculum Publishers
1510 Allegheny Drive
Colorado Springs, CO 80919
1-800-348-5646 (FIT LOGO)

Curriculum written BY teachers FOR teachers!

Sample Coordinate Grid



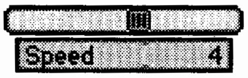
Visible Variables

Glen L. Bull and Gina L. Bull

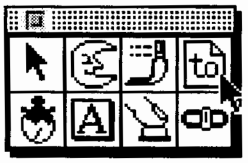
In *Mindstorms,* Seymour Papert described "microworlds" as "incubators for knowledge." A microworld was loosely defined as a computing environment that allowed learners to experiment with a particular set of ideas. Papert described a microworld in which learners could experiment with Newtonian physics, but he suggested that microworlds could be created for other areas as well.

The application *MicroWorlds* developed by Logo Computer Systems Incorporated (LCSI) combines Logo with hypermedia capabilities to allow learners to construct their own microworlds or experiment with ones developed by others. *MicroWorlds* combines almost all of the capabilities of *LogoWriter* with hypermedia buttons and text windows introduced in *HyperCard.* The names of objects sometimes differ slightly—a text window is referred to as a "text field" in *HyperCard* and a "text box" in *MicroWorlds,* for example—but the capabilities are similar.

MicroWorlds also introduces a new control called a "slider" not found in *HyperCard* or most other hypermedia programs. It is possible to create slider controls in *HyperCard,* but they are provided as built-in objects in *MicroWorlds.* Slider controls were initially introduced in LEGO DACTA *Control Lab,* a product jointly developed by LEGO DACTA and LCSI. Sliders in *Control Lab* allows users to control *LEGO* motors and gears connected to the computer. For example, a slider control on the computer screen might allow users to control the speed of a motor connected to the computer:



Slider controls in *MicroWorlds* allow users to control variables in Logo procedures. An input to a traditional Logo procedure can be created by placing the name of the variable on the title line. For example, a Logo procedure to draw a square could be created with an input named **Length.** An icon in the upper right corner of a Tools palette is used to access the *MicroWorlds* editor. Click once on the icon, which resembles a sheet of paper with the word "to" on it, to access the Procedures editor:

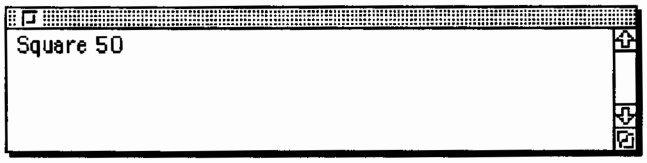


The editor can be used to create the following procedure for drawing a square. (Click the Editor icon on the Tools palette a second time to return to the previous *MicroWorlds* page when you have finished entering the procedure.) The variable name **Length** in the title line after the word Square indicates that the procedure has an input.

```
TO Square :Length
REPEAT 4 [FORWARD :Length RIGHT 90]
END
```

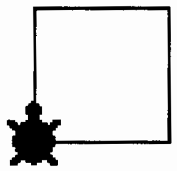
The colon in front of the word **Length** indicates that it is a variable—that is, a name that serves as a place holder for a value to be supplied later.

The user enters the name of a procedure in the *MicroWorlds* Command Center (shown in the following illustration) and presses the Return key to run the procedure. In the instance of the **Square** procedure, an input specifying the length of a side follows the name of the procedure. (If the input specifying the length is omitted, a message that says "Square needs more inputs" will appear.)



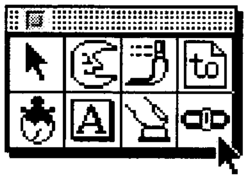
```
Square 50
```

After the name of the procedure is entered in the Command Center, the Logo turtle draws a square that is 50 turtle steps on each side when the user presses the Return key:



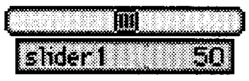
Sliders

In *MicroWorlds* a slider can be created to specify the length of a side in the **Square** procedure. The Tools palette contains a Slider tool in the lower right corner of the palette:

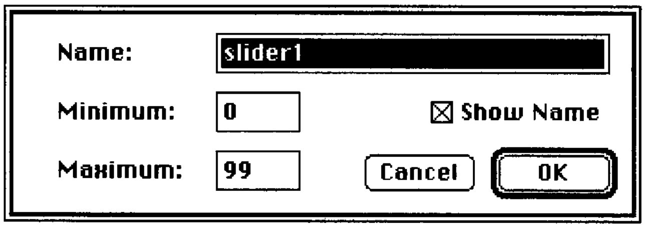


When the Slider tool is selected by clicking that square of the Tools palette, a hand holding a slider bar appears:

A slider is created by positioning the hand with the slider bar on the *MicroWorlds* page and clicking the mouse button once:

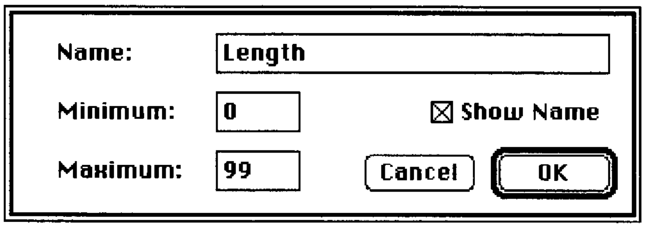


The initial name of the slider is **Slider1**, but a dialog box appears when the slider is created that allows its name to be replaced with any other name the user desires. The minimum and maximum range of numbers controlled by the slider can also be changed in this dialog box:



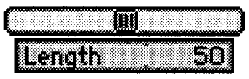
Name: slider1
Minimum: 0 ☒ Show Name
Maximum: 99 Cancel OK

In this particular instance the slider created was named **Length**:



Name: Length
Minimum: 0 ☒ Show Name
Maximum: 99 Cancel OK

The slider is renamed **Length** when the dialog box is closed by clicking the OK button:



After the slider has been created, the **Square** procedure can be changed so that the slider controls the length of each side in the square procedure. This adjustment involves two changes:

1. The input name **Length** is removed from the title line of the procedure (because the slider control will be used in place of an input to the procedure).
2. The name **Length**—without a colon—is substituted for the name **:Length**—with a colon—in the body of the procedure.

The revised procedure looks like this:

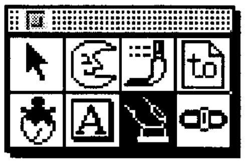
```
To Square
REPEAT 4 [Forward Length Right 90]
End
```

A subtle but important distinction is that in *MicroWorlds* the contents of a variable (such as the input to a procedure) are referenced by placing a colon in front of a variable name, while the setting of a slider is accessed by referring to the name of the slider, which has no colon in front of the name.

After a slider has been created and the **Square** procedure has been revised to incorporate its use, the size of the square will be controlled by the slider. As the *MicroWorlds* manual notes, use of a slider control has two benefits: First, the size of the square can be changed by adjusting the slider bar with the mouse, and, second, the current value of the variable is visible.

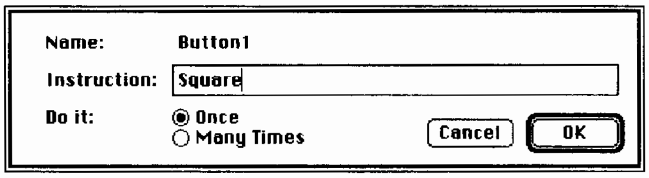
MicroWorlds Buttons

The revised **Square** procedure can be run by entering the name of the procedure in the *MicroWorlds* Command Center and pressing the Return key. *MicroWorlds* also provides the facility for creating a button that will produce a square when clicked. The Button tool is just to the left of the Slider tool on the *MicroWorlds* Tools palette:



After the Button tool is selected, a hand will appear:

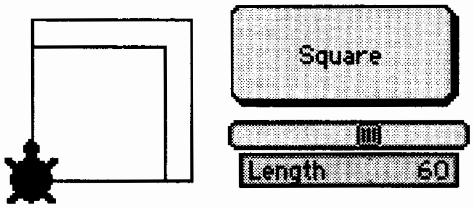
Use the hand to point to the location on the *MicroWorlds* page where you want the button to appear, and click the mouse button once. A dialog box appears. Enter the name of the procedure—in this instance, **Square**—that you want the button to run.


Name: Button1

Instruction: Square

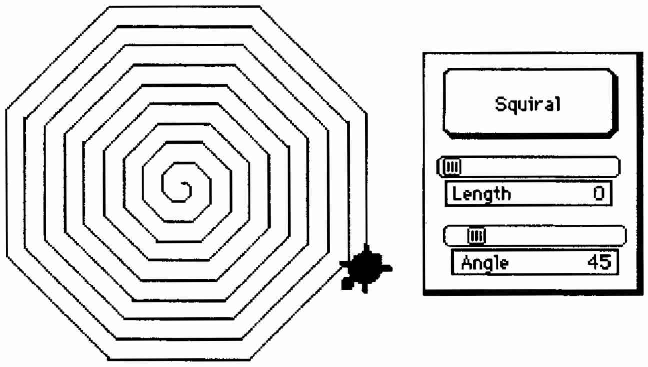
Do it: [✓ Once] Many Times

After the button has been created, clicking the button will run the procedure. In this instance, the slider **Length** controls the size of the square, and the button **Square** runs the **Square** procedure:



Experimentation With Variables

The addition of new controls such as sliders makes variables visible and encourages experimentation. **Squiral** is a well-known Logo procedure that causes the turtle to recursively spiral around a center point, increasing the length of each successive line it draws. In this particular instance an **Angle** slider controls the angle at which the turtle turns, hence, the shape of the resulting figure:



The **Squiral** procedure moves the turtle forward by the amount of the length specified in the **Length** slider, and then turns the turtle by the amount specified in the **Angle** slider:

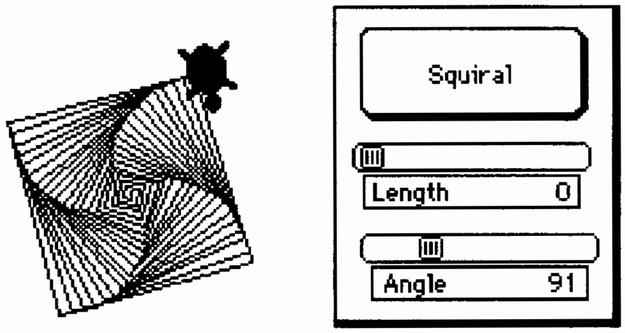
```
TO Squiral
Forward Length Right Angle
If Length > 75 [STOP]
SET "Length "Value Length + 1
Squiral
End
```

The **Squiral** procedure then increases the value of the **Length** slider by one and runs the **Squiral** procedure again. The line of code

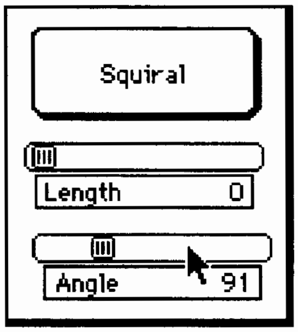
```
SET "Length "Value Length + 1
```

says to set the slider named **Length** to the current value of the **Length** slider plus one more. It is important to follow the syntax exactly in these lines; if there is only one quotation mark at the beginning of a name, don't add a second at the end when you enter the name.

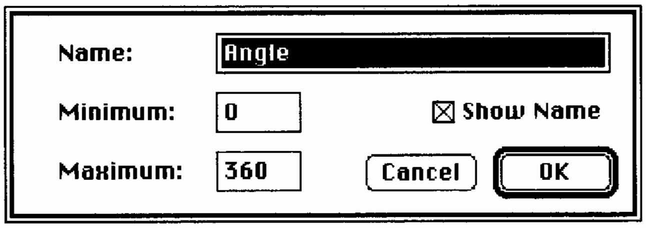
If the **Squiral** procedure begins with a starting angle of 91 instead of 45—as in the previous figure—an entirely different kind of spiral emerges. (*Troubleshooting Tip*: The **Squiral** procedure stops when the length exceeds 75 turtle steps. If the turtle doesn't move when the **Squiral** button is clicked, set the **Length** slider to 0 before clicking the **Squiral** button to start the procedure.)



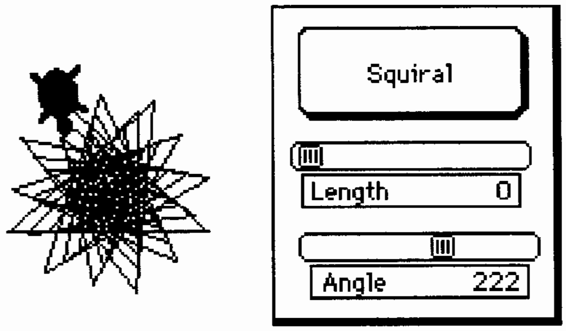
A beginning angle of 222 produces a different kind of figure. The default maximum value for sliders is 99, so it will be necessary to increase the maximum for the **Angle** slider before larger angles can be set. To modify the maximum angle of the **Angle** slider, hold down the Shift key on the keyboard and double-click the **Angle** slider:

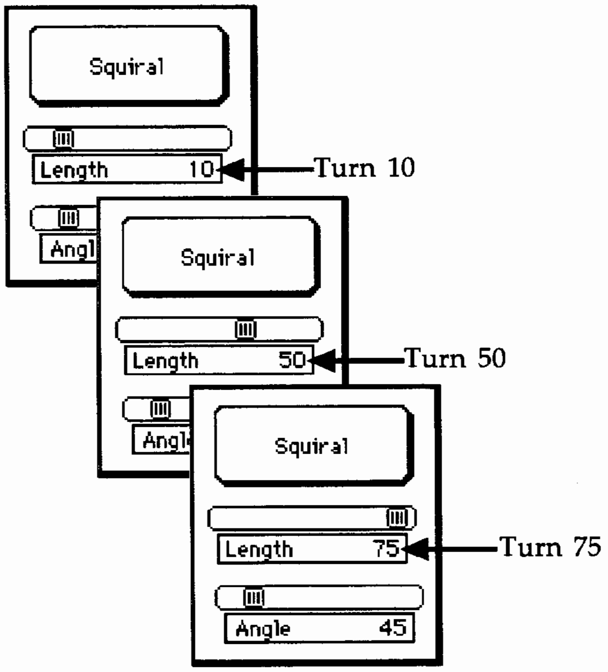


A dialog box for the **Angle** slider will reappear when the slider is double-clicked as the Shift key on the keyboard is depressed. Enter a maximum value of 360 in the Angle slider dialog box. Then click the OK button to close the dialog box.



When the **Squiral** procedure is run with an angle of 222 degrees specified, the following figure results:





In each instance the user uses the sliders to select a starting length and angle and then clicks the **Squiral** button to draw the figure. This process retains many of the strengths of traditional versions of Logo but adds newer hypermedia capabilities that have been emerging in recent years.

Dynamically Changing Variables

There is another aspect of visible variables provided by slider controls that may not be obvious on the static page. As the turtle draws each successive leg of the figure with a length of one greater than the previous line, the slider bar for **Length** gradually moves from 0 to 75 as the figure progresses. By the time the turtle draws the 10th line of the figure, the slider bar has moved a similar amount across the slider control. Similarly, when the turtle reaches its 50th turn, the slider bar has progressed a similar distance.

This process provides a visible window displaying changes in the variable as the procedure progresses.

Variables are a core concept in both mathematics and programming. They are extremely valuable in both disciplines but are sufficiently abstract that it can be difficult for beginners to grasp the underlying concept. By providing a method for making variables visible, *MicroWorlds* increases the likelihood of experimentation and exploration.

By an interesting coincidence, turtle fanciers affectionately refer to some types of turtles as "sliders." Introduction of this new device, which can be used to describe and control the motion of the turtle, seems congruent with Logo in both spirit and name.

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.

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

Have Some Fun With Recursion ...

by John Gough

With Logo, once you have learned how to make the turtle draw, it is fascinating to explore ways of drawing polygons and spirals. This is often the way that "recursion" is introduced. A recursive procedure is one that uses its own name as a command inside itself. Remember the old joke ?

> Q: How do you keep a drongo occupied for hours? (Hint: a "drongo" is an Australian idiot.)
> A: Give him a piece of paper with "Please turn over" written on both sides.

Here, it is the idea of the two-sided message that is recursive. The message on one side refers to the message on the reverse side, which refers to the message on the front, and so on, repeating, or recurring, ad infinitum. These recursive procedures are powerful ways of building complex patterns by successive repetitions or incremental changes.

Here is a procedure that repetitively makes a rosette pattern:

```
to rosette
repeat 4 [forward 80 right 90]
right 37
rosette
end
```

Hide the turtle, then type

```
rosette
```

In *LogoWriter* you will need to press the Stop key to stop it, or, like the drongo, it goes on forever.

The next procedure shows a different way to use recursion:

```
to grow.steps :length
ht
forward :length
right 90
forward :length
left 90
grow.steps :length + 2
end
```

The line before the **end** increases the value of **length** by adding 2 each time **grow.steps** repeats. To make the procedure work, type **grow.steps** followed by some number. For example, try typing

```
grow.steps 5
```

in the Command Center and press Return. This kind of geometrical recursion offers a great mathematical challenge, using concepts of distance and angle. But recursion can do much more than make intricate drawings and patterns.

And Now ... A Word About Embedded Recursion

Type the following procedure:

```
to bore.1
print [Hello]
bore.1
end
```

Now type **bore.1** in the Command Center. The word "Hello" will be printed endlessly unless you press the Stop keys. Of course, this is very boring. But to make recursion do useful things, we need to find ways of controlling it. Here is an example that does the same work as **bore.1**, but the **label** command tells the bored user what to do to alleviate the boredom and then **key?** responds when a key is pressed:

```
to bore.2
if key? [print[Whew!] stop]
print [Hello]
pu
setpos [-50 - 50]
ht
label [Press a key to stop me]
bore.2
end
```

The command **if** in the line containing **if key?** checks whether a key on the keyboard is being pressed. If so, the commands in the square brackets are run, which stops the procedure **bore.2**. Otherwise, the procedure moves to the next line and prints the word "Hello."

Once you see what **bore.2** can do, add **tone 200 20** between the **bore.2** line and the **end**. Run **bore.2** again. What happens? What controls the number of times it happens? This is an example of what can be called "embedded recursion." Every time **bore.2** runs, the line after the recursive call is never reached because the command **bore.2** intervenes and "loops" back on itself. When a key is pressed, the command **stop** does stop the next **print** line from running. But each time **bore.2** ran previously, it was left unfinished because the **tone** command was never reached. Thus, there is one tone for each "Hello" that has been printed except the last one, which was interrupted by **stop**. If this seems hard to understand now, don't worry. Embedded recursion becomes clearer when you see more examples and begin to realize how powerful it can be.

What else can we do with recursion? We can do many different kinds of drawing and animating. We can count numbers. We can build lists. We can explore the screen randomly, or roll dice. Here are some examples.

Recursion With Music

This first procedure slides up the musical scale:

```
to sing.up :jump
tone :jump 1
sing.up :jump + 100
end
```

Try

```
sing.up 1
```

and

```
sing.up 5000
```

You can control this procedure by inserting

```
if :jump > 9000 [stop]
```

If that's too painful, change 9000 to 1000.

Here's another example with music:

```
to sing.up.1 :step
tone 50 + :step 1
sing.up.1 :step + :step
end
```

Can you make a procedure that will sing down? How can you control such a procedure?

Recursion With Graphics

Controlling recursive procedures is very important. Many different kinds of controls are possible, but all of them depend on a line that uses **if** to test whether or not some condition is true. For example, we can control **grow.steps** by stopping it when **length** exceeds a specified size. Try inserting

```
if :length > 20 [stop]
```

at the beginning of the procedure. Of course you can change the size to make **grow.steps :length** recur more times or fewer times.

It is harder to control the **rosette** procedure we wrote earlier. Because it is built out of a repeating rotation of 37 degrees, we can use successive multiples of 37 to specify when **rosette** should finish. For example, insert

```
if heading = 333 [stop]
```

This will give eight repetitions.

Alternatively, we can use a variable that will successively count the number of times **rosette** has been repeated. To do this we need to initialize the counting variable in a separate procedure that will start **rosette** going:

```
to begin.rosette
make "occurrence 0
rosette
end
```

Then to make the counter work and test the number of occurrences, we insert the following lines in **rosette:**

```
make "occurrence :occurrence + 1
if : occurrence > 27 [stop]
```

This will give 28 repetitions of rosette. Can you see why? Remember, we started counting at zero.

Incidentally, it is important to place a controlling line correctly. If a controlling line is placed after the command that makes the procedure recur (the recursive call), the procedure will always repeat before it reaches the controller. Thus, the controlling line will never have a chance to work.

Other Effects

Recursion is well known for creating geometric patterns, tesselations, spirolaterals, and fractal diagrams. Often the programming involved is quite advanced and hard for beginners to understand. But simpler kinds of recursion can be used to get a feel for the ideas—and to have fun.

The following procedures use recursion to produce animation-like effects:

```
to square
repeat 4 [forward 50 right 90]
end

to roll.square
ht
pd
square
pe
square
right 10
roll.square
end
```

Or you can try this variation:

```
to rotate.square
ht
square
clean
right 15
rotate.square
end
```

Can you rotate it more quickly? Can you make it rotate as it drifts up toward the right-hand corner?

Here is a simple flashing device:

```
to label.flash
ht
label [flash]
tone 100 2
label.flash
end
```

Because the **label** command occurs in exactly the same position, the labeled word is deleted every second time it is put on the screen. (Note: You must use **px** to produce this effect on the Macintosh.)

This next set of procedures animates a flashing label. If its works too quickly, make the tone play longer by changing 1 to 5 or 15.

```
to scroll.message
pu
setpos [0 - 80]
seth 5
ht
send.message
end
```

```
to send.message
label [watch this space]
tone 100 1
label [watch this space]
forward 15
send.message
end
```

You can control **send.message** by inserting the line

```
if ycor > 80 [stop]
```

or

```
if xcor > 120 [stop]
```

This tests whether the value of the y-coordinate or the x-coordinate of the turtle has exceeded some specified amount. Sometimes if the turtle moves too far between one test and another it will actually exceed the specified value, but in doing so the turtle will have "wrapped" around the screen again. When this happens the actual x-coordinate or y-coordinate will then be smaller than the specified amount, and the test will fail to stop the recursion as desired.

To see the effects of a label being incorrectly deleted, try changing the second message to "watch this space" or "Watch this space." (Macintosh users need to use **px label** instead of using **label** twice.)

Instead of scrolling flashing labels around the screen we can stitch a row of designs across the screen. The starting procedure sets the initial position and then hands over to the procedure **button,** which is recursive:

```
to start.button
pu
setpos [-130 -80]
seth 60
setsh 12
button
end
```

```
to button
pd
stamp
pu
forward 20
button
end
```

You can control this by using a test of the value of the turtle's x-coordinate or y-coordinate, (**if xcor...**, or **if ycor...**), or testing whether the turtle has already stamped in that position (**if colorunder = 1 [stop]**).

We can develop these ideas to have more than one turtle trundling around the screen:

```
to start.trundle
set.up.1
set.up.2
trundle
end
```

```
to set.up.1
tell 0
setsh 23
setc 4
seth 120
pu
setpos [-130 80]
st
end
```

```
to set.up.2
tell 1
setsh 22
setc 5
seth 240
pu
setpos [130 80]
st
end

to trundle
if colorunder = 4 [stop]
tell 0
forward 15
pd
stamp
pu
tell 1
forward 15
seth random 360
tone 200 5
trundle
end
```

Can you make several turtles and labels move around the screen? Can you control it with an **if key?** test and use **print** and **ct** to give instructions about pressing a key to stop the recurring?

Here's a different way of moving around the screen:

```
to wind
seth 1
forward 20
wind
end
```

Experiment with different values for the heading. Try replacing the line

```
forward 20
```

with

```
forward 20
pu
forward 1
pd
```

How about this?

```
to wind.1 :heading :draw :up
seth :heading
pd
forward :draw
pu
forward :up
wind.1 :heading :draw :up
end
```

Try

```
rg
wind.1 5 2 4
```

or

```
wind.1 -5 4 2.
```

This example is as easy as falling off a ladder!

```
to topple
ht
forward 80
back 80
clean
right 5
topple
end
```

You can control the procedure by inserting

```
if heading = 90 [stop]
```

Can you make it topple to the left? Can you make a baton twirler? Here's a nice, active square dance procedure that runs randomly:

```
to dance
pu
setx random 130
sety random 90
dance
end
```

Try changing **pu** to **pd** or **px,** which will draw where there is nothing and erase where something has already been drawn. Can you make the color change randomly as the pen draws? Can you change it so that the dance occurs all over the screen and not just at the top right? Can you make a non-"square" dance, using randomly chosen inputs for **setpos**?

Recursion and Text-Mode Commands

Part of learning about recursion involves learning different ways of controlling recursion and different ways of using recursion to do interesting, intriguing, or just plain funny things. The more kinds of fun beginners can have with simple kinds of recursion, the better they will be able to use recursion powerfully. Here is a way to use recursion with text-mode commands such as **print, cf, cb, insert, cu,** and **cd** as a horizontal text slider:

```
to tab.over :times
repeat :times [tab]
print [move over]
wait 5
ct
tab.over :times + 1
end
```

Try

```
   tab.over 1
```

or

```
   tab.over 3
```

This can be controlled by inserting

```
   if first cursorpos > 120 [stop]
```

which tests whether the x-coordinate of the cursor's position is greater than 120. We can also have a text message placed at a random height:

```
   to shove.down
   if last cursorpos < (80 - random
      160) [print[write it here] stop]
   print[ ]
   shove.down
   end
```

Can you combine **tab.over** and **shove.down** to place text at random positions on the screen?

Recursion and Numbers

Recursion can be used to do much more than drawing, tessellating, moving text and labels, and animating. The next procedures count numbers and build a list:

```
   to start.counting :start
   continue.counting :start []
   end

   to continue.counting :start
      :number.list
   print :number.list
   continue.counting :start + 1
      sentence :number.list :start
   end
```

The input **number.list** is initialized in **start.counting** as an empty list when **continue.counting** is called. Each time **continue.counting** is called, **sentence** adds the value of **start** to the end of the list **number.list**. Try controlling this procedure by adding

```
   if :start > 25 [stop]
```

or

```
   if last :number.list > 25 [stop]
```

Can you make a set of procedures to start at any number you choose, count back to zero, and then stop?

The next example will make a jumbled list of numbers chosen randomly from 1 to 99:

```
   to start.jumble.numb
   jumble.numb []
   end

   to jumble.numb :number.list
   print :number.list
   jumble.numb sentence 1 + random 100
      :number.list
   end
```

You can control this recursion by inserting the instruction

```
   if  (count  :number.list)  =  20
      [print[Here's 20 jumbled numbers]
      stop].
```

Can you make a controlling line that will stop the procedure when a particular number is placed in **:number.list?** Can you change this so that the randomly chosen number is different from any previously chosen number? This is tricky. You need a line that will test whether the selected random number is already a member of the list of selected numbers. If the number has already been selected, the test loops back to **jumble.numb :number.list** without adding the random number; but if it hasn't, the test adds in the new number and then proceeds to **jumble.numb :number.list**. Try using another variable to "hold" the value of the new random number until it has been checked as a possible member of the existing list of selected numbers.

Here's a probability experiment that tosses a coin and counts the number of times a head appears:

```
   to toss.coins
   make "heads 0
   print[Press a key to stop]
   toss []
   end

   to toss :times
   if 1 = (random 2) [make "heads
      :heads + 1]
   if key? [(print[I tossed] :times
      [times, and got] :heads [heads])
      (print[That's a percentage of]
      100 * (:heads/:times)) stop]
   toss :times + 1
   end
```

How could you change this so that heads is twice as likely to come up as tails?

This next one is fairly obvious, because **butlast** knocks the end off the list called **bottles**:

```
   to wall.bottle
   pr [There were ten green bottles,
      standing on the wall ...]
   green.bottle [1 2 3 4 5 6 7 8 9 10]
   end
```

```
to green.bottle :bottles
print :bottles
green.bottle butlast :bottles
end
```

This will stop by itself with an error message when all the bottles are gone and **butlast** tries to knock the end off an empty list. To avoid this, try these tests:

```
if :bottles = [] [print[all gone] stop]
```

or

```
if empty? :bottles [print[no more]
   stop]
```

or

```
if (count :bottles) = 0 [stop]
```

Each of the three test lines do exactly the same work but use different techniques. Can you make a recursive procedure that sings the song "Twelve Green Bottles" or another counting song, such as "Five Little Ducks"?

Here is another probability experiment:

```
to start.letter.pick
letter.pick [ a b c d e f g h i j k l
  m n o p q r s t u v w x y z] [] []
end

to letter.pick :alpha :letter.list
   :choice
if member? "q :letter.list
   [print[Got a q!] stop]
make "choice item 1 + random 26
   :alpha
make "letter.list sentence
   :letter.list :choice
print :letter.list
letter.pick :alpha :letter.list
   :choice
end
```

Can you change this so that no letters are repeated in **letter.list**? Consider how the variable **choice** could be used to "hold" a randomly chosen item from **alpha** so that it can be tested to see whether that item was already a member of **letter.list**.

And Some More Graphics

Try experimenting with the next procedure to see how wide a line is:

```
to stripe :gap
forward 180
pu
right 90
forward :gap
left 90
pd
stripe :gap
end
```

Try typing

```
stripe 5
```

then

```
stripe 4
```

then

```
stripe 3
```

then

```
stripe 2.5
```

and so on. And try this one where the gap increases between the stripes. When do lines stop overlapping? Can you make a version of this procedure that counts how many times it has repeated or tells you how big the gap is now?

```
to stripe.1 :gap
forward 180
pu
right 90
forward :gap
left 90
pd
stripe.1 :gap + .1
end
```

Here's another interesting graphics example:

```
to chameleon :it
setc remainder :it
wait 5
chameleon :it + 1
end
```

Try replacing the second line with these commands:

```
setc :it
tone 200 5
```

Or try using

```
setsh :it
tone 100 5
```

so that the turtle's shape will successively be shape 1, shape 2, shape 3, and so on. You can use an **if key?** test to let whoever uses it stop the procedure when it reaches any shape the person chooses:

```
to streak
tell [0 1]
seth 180
st
tell [0 3]
pd
tell [1 2]
pe
tell all
forward 10
streak
end
```

or

```
to streak.1.start
tell [1 2]
pu
forward 40
tell all
px
seth random 360
streak.1
end

to streak.1
forward 20
streak.1
end
```

Can you insert a line to stop this when the user presses a key? Or after streaking some number of times, perhaps as many as the user specifies? Or after an amount of elapsed time? Or when some random event occurs? Can you extend this so that two pairs of turtles head in different directions as they streak? Can you make this stop when one pair collides with another? Can you make the streaking turtles "bounce" off the edges of the screen or some other spatial boundary?

It should now be clear that the only limit to the possible uses for recursion is your imagination. A few simple commands can do many kinds of powerful work.

John Gough
Deakin College
Toorak Campus
336 Glenferrie Road
Malvern, Victoria 3114
Australia

Statement of Ownership, Management, and Circulation (Required by 39 U.S.C. 3685). 1. Publication Title: Logo Exchange. 2. Publication No.: 0888-6970. 3. Filing Date: October 5, 1994. 4. Issue Frequency: Quarterly. 5. No. of Issues Published Annually: 4. 6. Annual Subscription Price: $34.00. 7. Complete Mailing Address of Known Office of Publication: ISTE, 1787 Agate St., Eugene, Lane, OR 97403-1923. 8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher (Not Printer): ISTE, 1787 Agate St., Eugene, OR 97403-1923. 9. Full Names and Complete Mailing Addresses of the Publisher, Editor, and Managing Editor: Publisher—International Society for Technology in Education, 1787 Agate St., Eugene, OR 97403-1923; Editor—Sharon Yoder, 1787 Agate St., Eugene, OR 97403-1923; Managing Editor—David Moursund; 1787 Agate St., Eugene, OR 97403-1923. 10. Owner: International Society for Technology in Education, 1787 Agate St., Eugene, OR 97403-1923. 11. Known Bondholders, Mortgages, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bondholders, Mortgages, and Other Securities: None. 12. 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. 13. Publication Name: Logo Exchange. 14. Issue Date for Circulation Data Below: Summer 1994 (Volume 12, No. 4). 15. Extent and Nature of Circulation. Average No. Copies Each Issue During Preceding 12 Months. 15a. Total No. Copies (Net Press Run): 1,403 15b. Paid and/or Requested Circulation. (1) Sales Through Dealers and Carriers, Street Vendors and Counter Sales: N/A. (2) Paid or Requested Mail Subscriptions: 1,174. 15c. Total Paid and/or Requested Circulation (Sum of 15b(1) and 15b(2)): 1,174. 15d. Free Distribution by Mail (Samples, Complimentary, and Other Free: 133. 15e. Free Distribution Outside the mail (Carrieres or Other Means): 50. 15f. Total Free Distribution (Sum of 15d and 15e), 183. 15g. Total Distribution (Sum of 15c and 15f): 1,357. 15h. Copies Not Distributed. (1) Office Use, Leftovers, Spoiled: 46. (2) Returns from News Agents: N/A.15i. Total (Sum of 15g, 15h(1), and 15h(2)): 1,403. Percent Paid and/or Requested Circulation: 87%. 15. Extent and Nature of Circulation. Actual No. Copies of Single Issue Published Nearest to Filing Date. 15a. Total No. Copies (Net Press Run): 1,350. 15b. Paid and/or Requested Circulation. (1) Sales Through Dealers and Carriers, Street Vendors and Counter Sales: N/A. (2) Paid or Requested Mail Subscriptions: 1,161. 15c. Total Paid and/or Requested Circulation (Sum of 15b(1) and 15b(2)): 1,161. 15d. Free Distribution by Mail (Samples, Complimentary, and Other Free: 91. 15e. Free Distribution Outside the mail (Carrieres or Other Means): 0. 15f. Total Free Distribution (Sum of 15d and 15e): 91. 15g. Total Distribution (Sum of 15c and 15f): 1,252. 15h. Copies Not Distributed. (1) Office Use, Leftovers, Spoiled: 98. (2) Returns from News Agents: N/A. 15i. Total (Sum of 15g, 15h(1), and 15h(2)): 1,350. Percent Paid and/or Requested Circulation: 93%. 16. This Statement of Ownership will be printed in Volume 13, No. 1 issue. 17. I certify that the statements made by me above are correct and complete. Patti Van Ordstrand, Distribution/Circulation Manager, October 5, 1994.

Papert Revisits "Powerful Ideas"

by Douglas H. Clements and Julie S. Meredith

The subtitle of Papert's famous book *Mindstorms* is "Children, Computers, and Powerful Ideas." What image would he wish to convey by this title today?

A short time ago, at a conference in Singapore, Papert presented the opening keynote address, the title of which, "Rethinking the Role of Technology in Education," had been assigned to him. He told the participants, "You've got it the wrong way around. We should be rethinking education in the context of technology." In the 1970s and 1980s, putting microcomputers in the hands of children was a revolutionary concept. The enabler was usually a teacher with a vision—a vision of breaking kids out of the traditional boundaries of school. However, today, most computers in schools are used to reinforce and improve what's always been there. Papert isn't exaggerating to make a point—research indicates that this is exactly what happens.

The reaction in Singapore was that this perspective was somewhat arrogant. Why should technologists decide what's important in education? Papert believes that this argument needs to be addressed more directly. It's what goes on in school that determines what's done with technology. Usually, we only call something a "technology" if it's recent; but paper, pencils, textbooks, and mathematical symbol systems are all technologies of earlier ages.

Paport recalled an example from *Mindstorms*: Children all over the world learn to recognize a parabola, draw one, and memorize its formula. The parabola, however, represents about one-billionth of today's mathematics. Why, then, is it taught so consistently? A big part of the answer is that it's easy to teach with pencil and paper. The absurdity is that, today, people continue to develop computer programs to draw parabolas much faster. But if you don't question why the parabola is the piece of mathematics that is selected for study, doing that with computers will just entrench traditions more deeply.

Another example harks back to number bases from the "new math" days. Number bases, like parabolas, are indeed wonderful things. However, why study them specifically? In the days of paper and pencil, kids spent a long time writing numerals on squared paper. This became a preoccupation. So, when people asked, "What are the concepts behind mathematics (arithmetic)?" they discussed what underlies this written system. The point is that there is one special condition that led to this perspective: written symbol systems. Thus, the very idea of the direction to take was dictated by the technological environment of that time. Therefore, today we have a curriculum overflowing with arithmetic manipulations of multidigit numbers.

Papert stated that he is interested not in the practical but the theoretical: What are the powerful ideas of mathematics? Not number bases. One of the powerful ideas he does acknowledge is the idea of feedback—a theme developed in some of his current work. For example, how would you get a robot turtle to walk around a table? One way to accomplish this is to use measurement and typical Logo commands. However, this method often doesn't work—for one thing, the turtle frequently jams against the side of the table. A better way is to use sensors that cause one response if the turtle touches the side of the table and another response if it doesn't. Feedback is a powerful idea, especially because of its generality. It is also powerful because it is body syntonic and it connects to so many other things. It is epistemologically powerful because it breaks away from the idea that knowledge is always exact.

Papert recalled that Lady Lovelace said that computers only do what they're programmed to do. There is a deep ambiguity in her statement that's worth pondering. Was the turtle programmed to go around the table? On the one hand, yes. But on the other hand, it was programmed merely to respond to feedback. The turtle can go around many tables, as well as other objects.

Papert returned to the issue of arithmetic. Can feedback even be relevant here? Think of finding the square root of 10. You can try to remember that algorithm—something with "20s" in it ... that's the most I can recall! Or you can home in: Try 2; 2 squared is 4; 4 squared is 16. Try 3; 3 squared is 9; that's close. And so on. This homing in is quite like programming the turtle to walk about the table by using feedback. Papert contends that there's not a problem in high school that you can't solve better by using this method of homing in. Using successive approximation, students get a real feeling for the mathematics. And it doesn't matter how the problem is written. Feedback can help you think about the orbit of the planets or the path of a moth flying around a candle.

In addition, feedback doesn't belong to mathematics alone. We need to break down barriers between subjects. Thinking in this way gives us a theoretical framework for thinking about different organizational structures of knowledge. We see many ways of organizing knowledge for the learner. Some use a hierarchical approach. One section of mathematics involves arithmetic, one section involves whole numbers, and one section involves multiplication of whole numbers. Down and down and down we go until somewhere somebody is working on one little piece. For some educators, that's structure. For Papert, it is chaos, because for the kid who doesn't see the connections, it *is* chaotic.

What's a better approach? Projects, for one. For example, one class wrote a *LogoWriter* program on the life cycle of the fruit fly. They had to understand the fly to make a presentation. The computer context made the project more exciting and easier to do.

Another class wrote computer programs to teach other students about a topic. One group of four kids created a project on the anatomy of the slug. They also had to produce all the documentation that went with it, including directions for using the program and a justification for the project's existence. For example, why should someone study the anatomy of a slug? Why is a project on a slug better than one on a frog? The kids pointed out that a frog is made up of obvious parts. Not so with a slug. It is hard to even know what is a part of a slug!

To come up with that explanation, they had to think about the structure of animals. It is surprising that there is a similar structure in frogs and in people. Why? What do they have in common?

One weakness of these students' educational environment was that they wanted to reach out and get more information about this structure, but it was difficult to do. Technology could have helped more here also, but no appropriate technology was in place.

The project side of Logo has worked well. The powerful-idea side has not been as well articulated. We've been ineffective in making that explicit. One technique that has been tried is to embed the ideas in projects. For example, Papert gave examples of kids writing *MicroWorlds* programs to make something leap over a barrier. The kids needed something else, though. They needed to have a lot of information about jumping, and they needed a place to get it. They needed a technological "Constructopedia." For example, how do you represent a jump? One powerful and easy way is to increase the x-coordinate instead of going **forward 1** to "run." If you want to go up, you can increase the y-coordinate—which is zero if want it to just run, but then you have a bug—you just keep going up. Therefore, you need gravity. You can think of this as something that eats vertical velocity. Powerful ideas like this would be available to kids in the Constructopedia. (Note: This approach still doesn't solve such problems as getting the right knowledge, the right level of knowledge, and the right amount of knowledge to keep the project going without getting so much knowledge that it attenuates discovery.)

The great thing about Logo projects is that kids think about things. Some of the things they think about should be ideas themselves, such as feedback. The task of educators is to look for striking examples of an idea such as this. They should then have students work on projects directed toward expounding and exemplifying powerful ideas. With this approach, the notion of a powerful idea is the most powerful idea.

Finding ways to get kids thinking about powerful ideas is up to you.

Preparation of this material was partially funded by the National Science Foundation under Grant Nos. MDR-8954664 and MDR-9050210. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Douglas H. Clements, a 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. Through a National Science Foundation (NSF) grant, he has developed a K-6 elementary geometry curriculum, *Logo Geometry*, published by Silver Burdett and Ginn in 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. He is also coauthoring a new version of Logo (Turtle Math) for learning elementary geometry.

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. She is also designing and programming a new version of Logo for the NSF-funded Investigations project.

Douglas H. Clements and Julie Sarama Meredith
State University of New York at Buffalo
Department of Learning and Instruction
593 Baldy Hall
Buffalo, NY 14260
Internet: CLEMENTS@UBVMS.CC.BUFFALO.EDU

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, 800/336-5191.

Product (• Indicates an ISTE-published title.)	*Member Price*	*Nonmember Price*
• Introduction to MicroWorlds—A Logo-Based Hypermedia Environment	*22.45*	*24.95*
• Math Activities Using LogoWriter—High School Math (specify Mac, MS-DOS, or Apple II disk)	*18.85*	*20.95*
• Math Activities Using LogoWriter—More Investigations (specify Mac, MS-DOS, or Apple II disk)	*14.35*	*15.95*
• Math Activities Using LogoWriter—More Patterns and Designs (specify Mac, MS-DOS, or Apple II disk)	*16.15*	*17.95*
• Math Activities Using LogoWriter—Probability and Statistics (specify Mac, MS-DOS, or Apple II disk)	*12.55*	*13.95*
• Way of the Ferret—Finding Education Resources on the Internet	*22.45*	*24.95*
101 Ideas for Logo	*14.95*	*14.95*
Hands-On Internet—A Beginning Guide for PC Users	*29.95*	*29.95*
How the Internet Works	*24.95*	*24.95*
How to Use the Internet	*17.95*	*17.95*
NetPower—Resource Guide to Online Computer Services	*39.95*	*39.95*
Realizing the Information Future—The Internet and Beyond	*24.95*	*24.95*
The Children's Machine—Rethinking School in the Age of the Computer	*12.00*	*12.00*
The Internet Companion—A Beginner's Guide to Global Networking	*12.95*	*12.95*
The Mac Internet Tour Guide—Cruising the Internet the Easy Way	*27.95*	*27.95*
The PC Internet—Cruising the Internet the Easy Way	*24.95*	*24.95*
The Whole Internet—User's Guide and Catalog	*24.95*	*24.95*
The Windows Internet Tour Guide—Cruising the Internet the Easy Way	*24.95*	*24.95*
Zen and the Art of the Internet—A Beginner's Guide	*23.95*	*23.95*

Receive an additional 18% discount when ordering 10 or more of the same title of ISTE-published products.

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

Code LX1

ORDER

Quantity	Title	Member Unit Price	Nonmember Unit Price	Total Price

SUBTOTAL	
Subtract 18% 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 $4.00; If COD, add $4.50	+
TOTAL	=

PAYMENT OPTIONS

☐ **Payment enclosed.** Make checks payable 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 $4.00 for order processing—
P.O. not including $4.00 fee will be returned.

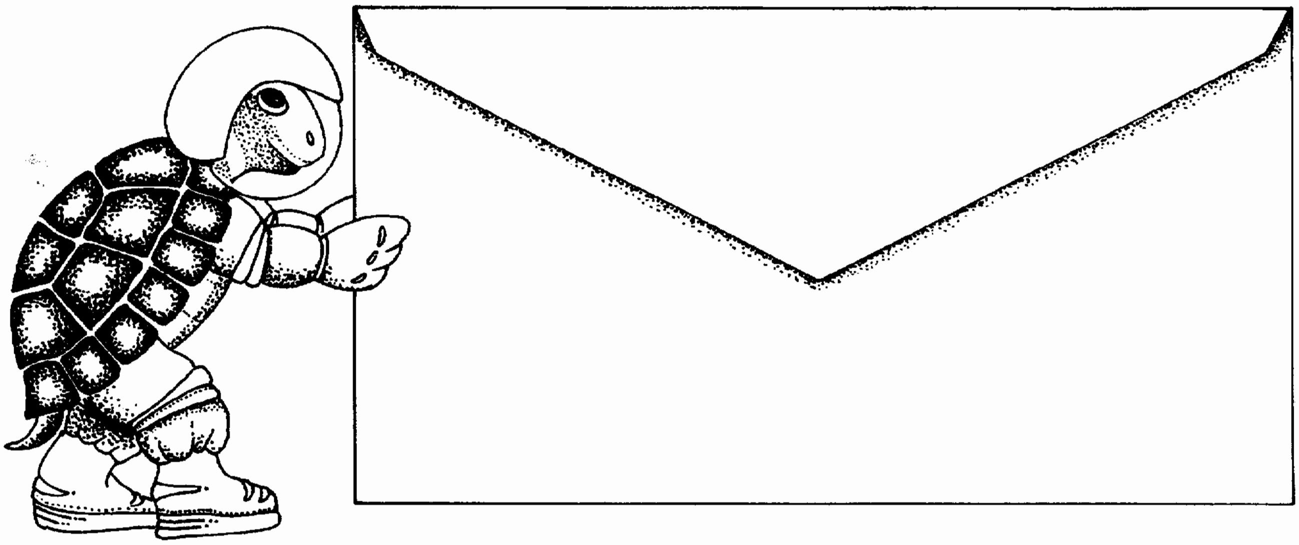
☐ **C.O.D.** for U.S. book orders only. You will pay UPS the total upon delivery by check or cash—
ISTE will add $4.50 order processing.

☐ **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.

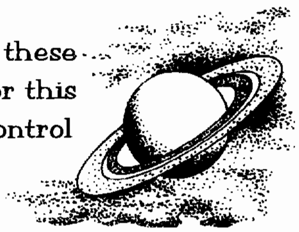
ISTE • 1787 Agate Street, Eugene, OR 97403-1923 • Order Desk 800/336-5191 • Fax 503/346-5890



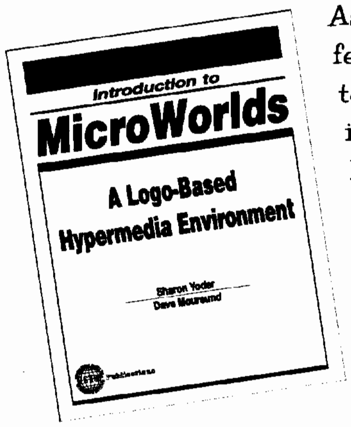
Push the envelope with MicroWorlds

Launch your students toward stimulating new classroom challenges and adventures. They can soar to a galaxy of new skills using MicroWorlds.

MicroWorlds is an exciting application that offers in a single program many of the features available in HyperCard, Kid Pix, and LogoWriter.



You're the pilot. Get ready now to grab the controls, blast off, and explore these exciting new worlds with your class! To help you prepare your crew for this fantastic voyage, Sharon Yoder and Dave Moursund back at mission control have developed a 222-page flight manual titled Introduction to MicroWorlds—A Logo-Based Hypermedia Environment.



As a computer application, MicroWorlds contains a wide range of features that make it easy and fun to work with color graphics, sound, text, and animation. As a programming environment, MicroWorlds includes a powerful and modern version of the Logo programming language.

It's time to broaden your universe. The countdown is underway. Have a nice trip!

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

Introducing the next best thing to two months off in the summer.

Turtle Math and MicroWorlds Math Links from LCSI

After years of development and consultation with teachers like you, LCSI introduces **Turtle Math** and **MicroWorlds Math Links:** two math tools for teachers who want to make math exciting.

Turtle Math and **MicroWorlds Math Links** provide a true advantage over any other math software. Each easy-to-use package provides students with an invaluable exploratory environment plus dozens of activities that help them think mathematically. So they learn more about math. And that means increased satisfaction for you.

Turtle Math, designed for students in grades 3 – 6, lets students use a collection of activities and challenges in which measurement and geometry is the context for exploring various math concepts.

Aimed at students in grades 4 – 8, **MicroWorlds Math Links** is an interactive learning environment that gives students concrete ways to explore abstract ideas and visualize answers to mathematical questions.

Both packages support the NCTM Standards, and are available for Macintosh computers.

If you're interested in exploring a new standard in math teaching tools, why not call us today for a free demo disk at:

1-800-321-5646.

LCSI®

And bring a little more sunshine into your classroom

ISTE BRINGS THE WORLD OF TECHNOLOGY CLOSER TO YOU.

By drawing from the resources of committed professionals worldwide, ISTE provides support that helps educators like yourself prepare for the future of education.

ISTE members benefit from the wide variety of publications, specialized courseware, and professional organizations available to them.

They also enjoy exciting conferences, global peer networking, and mind-expanding in graduate level distance education courses.

So if you're interested in the education of tomorrow, call us today.

International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923 USA
Phone: 503/346-4414 Fax: 503/346-5890
Order Desk: 800/336-5191
America Online: ISTE
AppleLink: ISTE
CompuServe: 70014,2117
Internet: ISTE@Oregon.uoregon.edu

WE'LL PUT YOU IN TOUCH WITH THE WORLD.