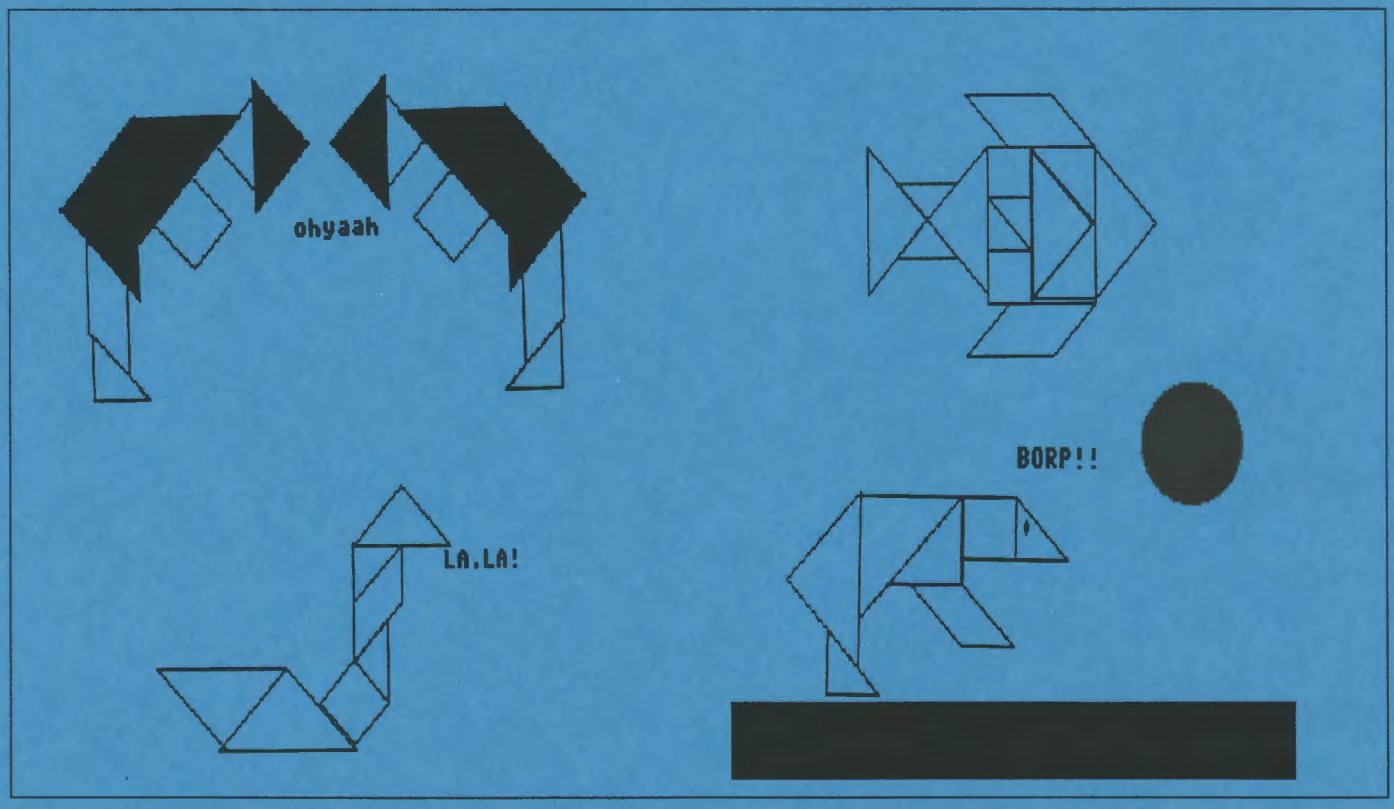
Journal of the ISTE Special Interest Group for Logo-Using Educators

LOGO EXCHANGE

March 1990 Volume 8 Number 7



International Society for Technology in Education

ISTE Publications

ef•fec•tive\i-'fek-tiv\adj (14c)

1 a : producing a decided, decisive, or desired effect b : IMPRESSIVE, STRIKING
2 : ready for service or action

Computer-Integrated Instruction: Effective Inservice

Dave Moursund's comprehensive series on inservice training for computer using educators has grown. ***Effective Inservice for Secondary School Mathematics Teachers*** and ***Elementary School Teachers*** are joined by texts for ***Secondary School Science Teachers*** and ***Secondary School Social Studies Teachers.***

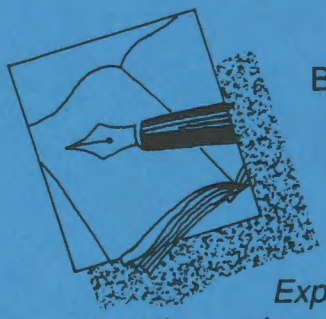
Based on a National Science Foundation project, these volumes bring you the latest research on effective training. Each work contains specific activities and background readings that enable you to hold inservices that result in positive, durable change at the classroom level.

If you design or run computer-oriented inservices, ***Effective Inservice for Integrating Computer-As-Tool into the Curriculum*** will help you develop a sound program through theory and practice. Sample forms for needs assessment and formative and summative evaluations are included.

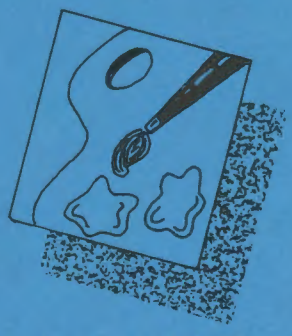
Each of the five volumes comes in a three ring binder that includes both hard copy and a Macintosh disk of the printed materials. Individual Math, Science, Social Studies, and Elementary School volumes are $40 each ($3.70 shipping per copy). Computer-As-Tool is $25 ($3.70 shipping per copy). The complete set of five is available for the discounted price of $150 ($7.50 shipping per set).

ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; ph. 503/346-4414.

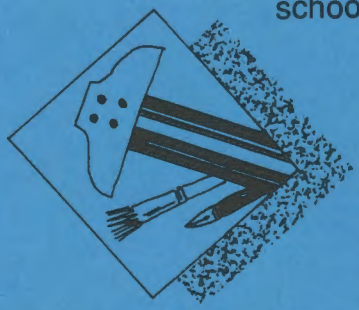
Make your desktop publishing software earn its keep.



By now you have discovered that there is more to desktop publishing than mastering the keystrokes and commands of the software. *Exploring Graphic Design* teaches you how to plan and produce letterhead, posters, newsletters, manuals and books.



Exploring Graphic Design is a concise and thorough overview of essential design principles and their application to practical problems. It complements any desktop publishing program. Perfect for secondary school or university classes, or as a helpful reference for adults.



Get your money's worth from your desktop publishing software by *Exploring Graphic Design.*
$9.95 plus $2.65 shipping

ISTE, 1787 Agate St., Eugene, OR 97403; ph. 503/346-4414.

ISTE

LOGO EXCHANGE

Volume 8 Number 7 Journal of the ISTE Special Interest Group for Logo-Using Educators March 1990

Logo Exchange is the journal of the International Society for Technology in Education Special Interest Group for Logo-using Educators (SIGLogo), published monthly September through May by ISTE, University of Oregon, 1787 Agate Street, Eugene, OR 97403-9905, USA; 503/346-4414.

POSTMASTER: Send address changes to Logo Exchange, U of O, 1787 Agate St., Eugene, OR 97403. Second-class postage paid at Eugene OR. USPS #000-554.

Contents

ISTE Membership

	U.S.	Non-U.S.
	28.50	36.00

SIGLogo Membership (includes *The Logo Exchange*)

	U.S.	Non-U.S.
ISTE Member Price	25.00	30.00
Non-ISTE Member Price	30.00	35.00

Send membership dues to ISTE. Add $2.50 for processing if payment does not accompany your dues. VISA and Mastercard accepted. Add $18.00 for airmail shipping.

From the Editor

Are You a "Language Chauvinist?"

This quarter I'm teaching a new course here at the University of Oregon called "Programming Languages for Educators." This course is designed to expose graduate students in computer education to important computer science concepts as well as to help them expand their knowledge of programming in at least two different programming languages. Because this is an education class, we will be discussing the issues surrounding the teaching and learning of computer programming while we ourselves are engaged in the process of teaching and learning.

I have been preparing for this course by rereading Bruce MacLennan's excellent book *Principles of Programming Languages* (Holt, Rinehart, and Winston). This book again struck me with the importance of viewing programming languages in the context of their history. While this book is indeed a computer science book aimed at teaching readers how to *develop* a programming language, it can easily be viewed at another level, providing the reader a deep understanding of why each major programming language or generation of programming languages has the characteristics that it does. When I first read the book some years ago I found that it put into perspective much of what I already knew about programming in a way that made me much more tolerant of the diversity of programming environments. As I reread it, I again find myself struck by the importance of understanding historical context, even in a field as new as computer science.

In the Logo community it is quite common to hear extremely disparaging remarks about other programming languages, especially BASIC. Those of us in love with Logo can't seem to imagine how anyone would want to program in any other language. If you talk to those outside of the Logo community who have different "favorite" programming languages, you'll find that they too think that *their* programming language is the only one to use, be it Pascal, C++, or Prolog.

Historically, however, each language or generation of languages has had its role to play. FORTRAN brought us away from assembly language programming and allowed us to use fairly standard mathematical notation. Algol laid the groundwork for most significant future programming languages. Pascal brought us simplicity and a move towards the needs of the applications programmer. Some of the early languages have been extremely successful; others have merely provided the foundation for future evolution. But each has made its contribution. It should be particularly interesting to *LX* readers that MacLennan's only mention of Logo is a reference to its influence on Alan Kay's development of Smalltalk. In this context, Logo is primarily viewed as having a small place in computer science history!

But what does all of this computer science "stuff" have to do with those of you using Logo in the classroom?

First and foremost, I think that it should remind us not to be "language chauvinists"; not to think that "our" language is the only language – or even that our version of Logo is the only one. Even your *LX* editor uses a variety of programming languages. I write programs in BASIC, Pascal, DBASE, Logo, and Hypertalk. Each has its place; each has its reason. Logo is not *always* the best tool for my needs.

More importantly, perhaps, let's be careful not to get stuck insisting that Logo not grow and change. Among some Logophiles there was a great deal of resistance to the LogoWriter interface when it was first introduced. Somehow it wasn't "real" Logo. I've heard quite a few people complain about some of the new features in LogoWriter and Logo PLUS. These detractors seem to think that Logo should remain pure to its traditional form. A proliferation of such attitudes will soon leave Logo as nothing more than a mention in the history of computer education. Educators will move on to more flexible and powerful tools that make use of the capabilities of new hardware and the needs of a changing population of computer using students.

And what of other software besides programming languages? In this issue we again have an article by Glen and Gina Bull on using *HyperCard*. Why? Because Glen and Gina (and many others) see Logo as part of a larger class of software that is "learner based"; software that puts the learner rather than the computer in control. No doubt you have noticed more articles along these lines in *LX* . These articles can help remind us all that it is generally the Logo environment that most of us value. For most of us, Logo is clearly more than just a programming language.

So where will the future take those of us who use Logo today? Hopefully towards better and better "learner based" environments; environments that don't control the student; environments that allow all users to make the computer a powerful tool in their lives. Let's hope that the developers of Logo versions continue to provide us with increasingly sophisticated versions of Logo that can compete successfully with the other wonderful software tools that are available for today's newest hardware.

Sharon Yoder, SIGLogo/ISTE
1787 Agate Street
Eugene, OR 97403

Monthly Musing

Do It Again!

by Tom Lough

During February and March of last year, I reported the responses of many readers to the question, "What is your favorite Logo command and why?" One of the most frequently mentioned commands was REPEAT. I got to thinking about that the other day. What about REPEAT would make it so popular?

The need for a REPEAT command is spontaneous and universal, as far as I can tell. One of the first questions I hear in a Logo workshop is "Isn't there some way to get the computer to do this again without having to type it [or scroll back up to it] each time?" We really seem to have this craving for an easy method of getting the computer to do the same thing over and over again. Molly Watt once told me that REPEAT was not included in the original MIT Logo version, but was added as a result of this expressed need. Hearing this makes me rather proud of REPEAT's special heritage.

Another special aspect of REPEAT is its ability to provide surprises. There is something heady about not being able to predict the outcome of several Logo commands repeated a bunch of times. I'm sure that nearly all LX readers have observed the astonishment and delight of a new Logo user who typed

```
REPEAT 500 [ several Logo commands
   here! ]
```

But REPEAT has some surprises in simpler (and more elegant) expressions, too. Remember an encounter similar to the following?

```
REPEAT 3 [FORWARD 50 RIGHT 60]
```

Oooops! Now, why doesn't that make an equilateral triangle?

I like the way REPEAT seems to invite experimenting with something resembling controlled conditions (related to the so-called scientific method). REPEAT makes it easy to change one aspect of a group of commands while keeping all others the same. For example, when students attempt to discover how to draw a five-pointed star, they often go through a process such as

```
REPEAT 5 [FORWARD 50 RIGHT 72]
REPEAT 5 [FORWARD 50 RIGHT 120]
REPEAT 5 [FORWARD 50 RIGHT 150]
REPEAT 5 [FORWARD 50 RIGHT 145]
```

whereupon they might stop, if the resulting figure seemed to be good enough. Varying the angle alone gives them some very special immediate feedback.

Here is a variation that has led to some interesting discussions about what affects size and what affects shape.

```
REPEAT 4 [FORWARD RANDOM 100
   RIGHT 90]
REPEAT 4 [FORWARD 100 RIGHT
   RANDOM 90]
REPEAT 4 [FORWARD RANDOM 100
   RIGHT RANDOM 90]
```

And, sooner or later, students get the idea of trying to put one REPEAT command inside another. This can lead to some really complicated results, especially if other Logo commands are sprinkled among the REPEATs.

```
REPEAT 5 [FORWARD 20 REPEAT 10
[RIGHT 38 BACK 40 REPEAT 4 [RIGHT 65
   FORWARD 70] LEFT 20]]
```

What about the one below? Care to make a prediction before you type this in?

```
REPEAT 1 [REPEAT 1 [REPEAT 1
   [REPEAT 1 [REPEAT 1
   [FORWARD 1]]]]]
```

Then try

```
REPEAT 2 [REPEAT 2 [REPEAT 2
   [REPEAT 2  REPEAT 2
   [FORWARD 1]]]]]
```

These aspects of REPEAT add up to a rather nice package. Once your students are hooked on REPEAT, then the stage is set for you to show them some really powerful stuff! I'll show you what I mean next month.

PS: What is the smallest value you can get REPEAT to accept for the number of repetitions of a list of commands and not produce an error message?

```
REPEAT 100 [ FD 1 ]
```

Tom Lough
Founding Editor
PO Box 394
Simsbury, CT 06070

Logo Ideas

"Driving" the Turtle

by Eadie Adamson

In the October 1989 issue of LX Diane Miller wrote about "The Turtle As Car," in which her student used a recursive procedure to move a car. Diane suggested that there were methods of steering a car as well. (See, for example, Tom Lough's Monthly Musings in May 1988, LX, page 3)

For the past several years I have been working with a group of boys developing motion games. One of their first tasks is to get the turtle moving as Diane's student did:

```
repeat 999 [forward 1]
```

proceeding next to a recursive procedure

```
to drive
forward 1
drive
end
```

and then to a recursive procedure with inputs.

```
to drive :speed
forward :speed
drive :speed
end
```

A turtle following a set path, however, did not satisfy my students for long. They knew they needed to control direction if they were ever going to adapt the procedure to a game!

How Can You Steer It?

There are several ways of adding "steering" to the simplest drive procedure, each with its own advantages and/or disadvantages. Here's one we used that does not require changing the original procedure.

First, we wrote procedures to output the cardinal directions, using numbers for headings:

```
to north
output 0
end

to south
output 180
end

to east
output 90
end

to west
output 270
end
```

Turning a turtle north, simply use seth north. For south, seth south. That much works fine for setting up a turtle to move, but how can you use this without changing drive? It's simple, actually!

Control Is the Answer

LogoWriter has 10 keys that can be programmed so that when the Control key is held down while one of these keys is pressed, a given command or series of commands or procedures will be activated, interrupting whatever else is in process on the screen. Programmed Control keys can even work independently of other procedures. Further, you can use them in the immediate mode as well, almost as if they were procedures. Puzzled? Read on....

The ten "programmable" keys are N O P Q R and V W X Y Z . To program a Control key use this form:

```
when "key [   here is where a task
   goes   ]
```

By typing an instruction such as

```
when "n [forward 50 right 90]
```

in the Command Center and then pressing Return, the "N" key is activated for as long as the computer remains on, or until the command **clearevents** is used. Pressing Control-N causes the turtle to go **forward 50 right 90.**

My students first explored this idea by activating the keys in the Command Center and then trying them out. However, a more efficient way of using these keys is to write a procedure that sets up the keys to be used. The idea that is sometimes hard for students to grasp is that if the instructions that define the actions of the Control keys are in a procedure, the *procedure must be invoked* to activate the keys. Simply writing a procedure on the flip side does not make the Control keys active.

The keys procedure can eventually go in a startup procedure on the page, but it is important to take the time to be sure the students understand that programming the keys alone is not what activates them. Spend some time setting up Control keys in a keys procedure, typing keys to activate the keys, testing them out, then typing **clearevents** to clear the keys, then running the keys procedure again to reactivate them.

With a "drive" and a "keys" procedure, the Control keys can be used to actually control the turtle's movement. What is fun for students in this context is perceiving the enormous extension this makes for their driving procedure without requiring them to reprogram **drive**. Somehow writing a second short procedure (or more) seems easier to many students!

Another Direction

One of the first things we tried when we first used LogoWriter was a project just like this. The first task was to create scenery, then get an object to move with a drive procedure. Instead of using Control keys for direction, however, we turned them into keys which changed the speed of the turtle and eventually made a kind of speedometer that reported the speed. How? Read on...

Changing Speeds

Students needed to have a drive procedure with inputs for the speed. Then I explained about the Control keys and showed them how to program a turtle to move faster. Pressing a Control key was used to alter the input to drive.

First, think about what occurs when something goes "faster." The drive procedure has an input, **speed**, that represents the "speed" in turtle steps. If "faster" is defined in terms of speed, you are increasing – adding to – the speed. The **speed** in the drive program is the input to forward, representing the number of turtle steps, or the distance, to be advanced each time the drive program is called.

Prior to working with Control keys, the method many students used to make their cars go faster was to stop the procedure and restart with a larger number for input. Faster, then, meant increasing the input for drive. How can this be expressed with a Control key? Like this:

```
when "z [make "speed :speed + 1]
```

What we are saying is: "Take the value of speed the turtle is currently using (a number) and add one to it. Call this the speed." We increment the speed by 1. This takes only a fraction of a second to take effect. Pressing Control and the letter z while **drive** is operating makes the turtle go faster; pressing the two keys again increases the speed again.

Slow Down or Stop!

Eventually the question arises: "How can I slow down the turtle?" There already is a model for making the speed faster. What makes a speed slower? Subtract instead of add! Most students have no difficulty figuring this out and choosing another key to program for slowing down the turtle.

Soon another question inevitably comes up: "How can I make the turtle stop without stopping the procedure?" Stopping means the speed is 0. We want to make the speed nothing, not less than the current speed or more than the current speed, but 0. That change can be expressed as:

```
make "speed 0
```

All we want is for the speed to be 0: nothing, no speed, not moving.

Once these three options are programmed, the other keys can be used for changing directions, changing "lanes" if the cars are on a highway, and so on. Some students wrote a "pass" procedure as well.

Speedometers

Earlier I mentioned speedometers. How can you get your program to tell you the "speed" each time a Control key is pressed to change the speed? Here's one way to handle the problem.

First, write a speed procedure. This procedure will display the speed in the Command Center, rather than on the screen. Since there may be commands in the Command Center, the procedure should clean that up first, then show the current "speed" of the turtle, the value of the variable speed. It might look like this:

```
to speed
cc
type:speed
end
```

(I prefer to use the command **type** here, rather than **show**. While **show** *will* display numbers without brackets, it poses a potential problem: if we add any text: the phrase will then appear in brackets. **type**,

Logo Ideas - continued

on the other hand, displays text without brackets. The fact that the cursor is left at the end of the line can be taken care of in two ways: either add one more line: **type char 13** (char 13 is the ascii value for the Return key) – or simply clear the commands as we did here each time the procedure was invoked.

Some students chose to print the speed on the screen. They needed to change the speed procedure to clear the text, **ct**, in order to avoid a string of numbers running down the left side of their scenery.)

More complex versions of a speedometer might use a multiplier and report a relative miles per hour:

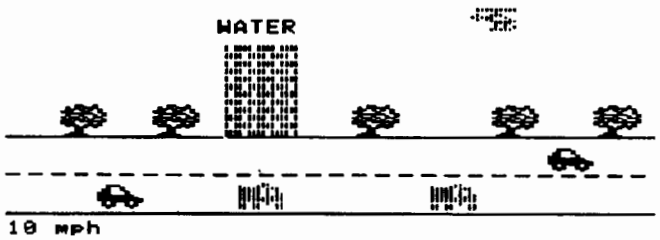
```
show sentence :speed * 10 "m.p.h.
```

This speed procedure is then used with each speed-altering Control key. Insert the word speed after the commands to change the value of speed. If there are three keys programmed to alter the speed, be sure to add the word speed just before the end bracket in each command. Here's an example:

```
when "z [make "speed :speed + 1
   speed]
```

Be sure to type the word **keys** (or whatever procedure name has been used to set up Control keys) before trying this. Simply making a change in the keys procedure on the flip side does not have any effect on the Control keys until the procedure is invoked again.

Type **drive 1** to begin. Watch the Command Center as keys are pressed to speed up and slow down or stop. Each time the speed is changed, the **speed** procedure will also be called into action. Voila! a speedometer!



Putting It All Together

The final important step is to write a superprocedure that sets up the keys and starts **drive**. Most of my students like to be able simply to type "go" and have things begin. Later the "go" procedure could also contain a setup procedure that put the vehicles into starting position. A simple "go" procedure would look like this:

```
to go
keys
drive 1
end
```

More ideas on driving at a later date. Meanwhile, enjoy the ride!

Eadie is a computer coordinator at The Allen-Stevenson School, an independent school for boys in New York City.

Eadie Adamson
1199 Park Avenue, Apt. 3A
New York, N.Y. 10128

Logo and Tangrams

by Lani Telander

As a teacher, I'm always looking for activities that will keep my students interested and my objectives accomplished at the same time. I have been doing a Tangram Unit with fifth graders for the past three years and each year I get more excited about the results.

My objectives for the Tangram Unit are to have the students work in a cooperative-learning activity, effectively using LogoWriter as a programming tool, and end up with a finished product of which they can be proud.

I begin by giving teams, usually two to three students to a team, the seven tangram puzzle pieces. They use the first couple of sessions to come up with a design the team agrees upon, using all seven pieces (some teams use more!) Each team then traces around the pieces on paper so they have their drawing from which to work. All of this work is done away from the computer.

The first step when the students begin working on the computer is to write a procedure for each tangram piece. This amounts to writing five procedures, since the tangram pieces consist of 1 square, 2 small triangles, 1 medium triangle, 2 large triangles, and 1 parallelogram. We use the following procedure for the square as the base upon which to write the other procedures:

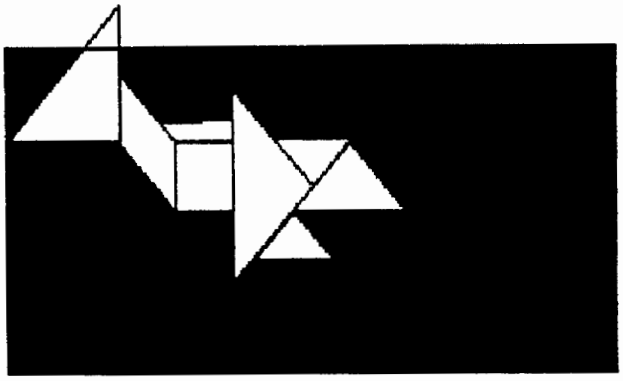
```
TO SQ
REPEAT 4 [FORWARD 30 RIGHT 90]
END
```

By using FORWARD 30 the tangram designs fit nicely on the screen without wrapping.

After the above procedures have been written, the challenging work of positioning the turtle begins before the next shape can be called up.

by Susan Bowers and Emilie Hitch

These fifth graders worked eagerly on this unit right up to the last session. They made use of the word processing feature of LogoWriter to write captions, TONE to enhance their designs with music, and, of course, color. In addition, this has been the most effective way for me to teach the Total Turtle Trip Theorem and SETHeading.



by Jason Grais and Geoffrey Nadler

This past year, the conclusion of our Tangram Unit coincided with Grandparents' Day at our school. We had the Computer Lab open so the fifth graders could share their work with their grandparents and parents. The enthusiasm and pride they showed for all their hard work was contagious!

Lani Telander
The Blake School
Highcroft Campus
301 Perry Lane
Wayzata, MN 55391

About the Cover

This month's cover shows the work of Lani Telander's students, as indicated below:

Ashleigh Cashman and Cassie Powell

Ben Mackay and Trevor Rusin

Robin Kreiser and Cindy Sher

Erin Clarkson and Andrew Steiner

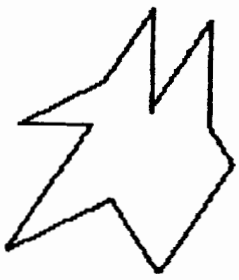
Beginner's Corner

Pencil & Paper or the Turtle?
by Dorothy Fitch

Have you noticed that sometimes the pictures that you plan are much easier to draw with a pencil and paper than to create using the turtle? And other times, it seems much easier to use the turtle than to attempt the same design with a pencil? This month, we'll take a look at different ways of approaching Logo designs.

Last month's column had a lot to do with stars. It reminded me of another stellar (and true) example of how there is often more than one approach to a problem in Logo.

Cathy, age 6, used Kinderlogo, a single keystroke Logo program in her first grade classroom once a week. The only commands available to her were F (to move the turtle forward 10 turtle steps), R (to make a 30 degree right turn), and L (to make a 30 degree left turn). Priscilla Flanagan, her teacher and Kinderlogo co-author, was present when Cathy created her star designs and related these fascinating observations to me.



With just three commands to use, Cathy drew this design one fall day. The intent was obviously a star and Cathy asked if her picture could be saved, although she didn't seem totally pleased with the result. But, her allotted time at the computer was up. We all know how frustrating that can be!

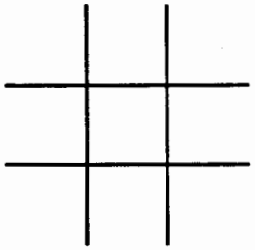
Later the same day, Cathy asked her teacher if she could have another turn at the computer. Although this was not normally allowed, her teacher sensed that this was somehow important and let her have some additional time. Cathy's next star looked like the star to the right:

What happened between the time Cathy drew the first star and the time, just a few hours later, when she drew the second star, we can only imagine. It is clear, however, that she must have spent some amount of time during the day thinking about the star and how she could improve on it. She must have known that she needed to use a different approach. Did the solution come in a flash, like a cartoon light bulb, or did she spend time doodling with a pencil, trying out possibilities? At some point she must have reached the conclusion that she should perhaps draw the star in Logo the way she would draw it with a pencil, not necessarily the way she'd seen stars drawn in books, with just the outline showing. Cathy was pleased with her second attempt at a star.

It really is quite exciting to see such concrete evidence of a thought process. Having witnessed this one isolated incident, it makes us wonder how often similar problem-solving occurs when Logo isn't there to provide "hardcopy."

Epilog: Cathy created this final star in the spring of the same year—a new variation on a familiar theme, and a neatly symmetrical one at that!

In this example, drawing the star in the same way you would with a pencil seemed to be the best approach. But that is not always the case in Logo. Consider a typical tic-tac-toe design, made up of just four straight lines.



If you were to draw this in Logo the same way you would with a pencil, you would have to pick the pencil up several times. This is the easiest way for humans, though not necessarily for turtles. Although in Logo you can determine the exact spots for placing the pen, your program might end up rather long, inflexible and hard to debug, like one of these rather gruesome examples:

```
TO TICTACTOE
PENUP
FORWARD 45
PENDOWN
BACK 90
PENUP
FORWARD 90
RIGHT 90
PENUP
FORWARD 30
LEFT 90
PENDOWN
BACK 90
PENUP
FORWARD 60
LEFT 90
PENDOWN
BACK 30
FORWARD 90
RIGHT 90
PENUP
BACK 30
LEFT 90
PENDOWN
BACK 90
END
```

```
TO TICTACTOE
PENUP
SETXY -15 45
PENDOWN
SETXY -15 (-45)
PENUP
SETXY 15 45
PENDOWN
SETXY 15 (-45)
PENUP
SETXY -45 15
PENDOWN
SETXY 45 15
PENUP
SETXY -45 (-15)
PENDOWN
SETXY 45 (-15)
END
```

These procedures work, but is either one the best way to draw it in Logo? Take a closer look at the tic-tac-toe design. Can you see any ways to repeat a pattern to draw the design? Remember that anytime that you can reduce a design to a repeated pattern, you gain a set of instructions that are:

- more efficient
- easier for you to debug or modify
- easier for others to read and understand
- more powerful and flexible
- more easily adapted for use in other projects that you develop

Here are some ideas for other ways to draw the tic-tac-toe design. You wouldn't choose any of these ways to draw the design with a pencil and paper, but they are rather suited to the turtle.

1. The C pattern

Picture the tic-tac-toe design made up of 4 squared-edged letter C's that look like this:

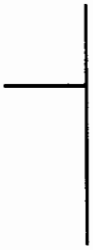
Just repeat the pattern four times for the tic-tac-toe design.

```
TO C
LEFT 90
FORWARD 30
BACK 30
RIGHT 90
FORWARD 30
RIGHT 90
BACK 30
FORWARD 30
END
```

```
TO 4C
REPEAT 4 [C]
END
```

2. The L shape design

In this version, the tic-tac-toe design is created using four double-L shapes, like this:

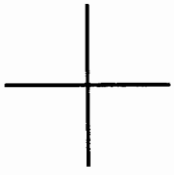


```
TO L
FORWARD 90
BACK 30
RIGHT 90
BACK 30
END
```

```
TO 4L
REPEAT 4 [L]
END
```

3. The letter T

Here is a version of the tic-tac-toe design that uses a repeated T-like shape:



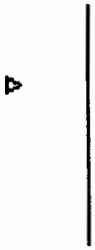
```
TO T
FORWARD 60
BACK 30
RIGHT 90
FORWARD 30
BACK 60
FORWARD 30
END
```

```
TO 4T
REPEAT 4 [T]
END
```

Beginner's Corner - continued

4. The jump, turn and draw method

Finally, this version causes the turtle to perform a pirouette midway through each repetition of the pattern. It goes forward, then hops over to the beginning point of the next line. It is similar to the L pattern above, but is actually a little quicker. Repeat the pattern four times to complete the tic-tac-toe design.



```
TO JUMP
FORWARD 90
LEFT 135
PENUP
FORWARD 42.43
PENDOWN
LEFT 135
END
```

```
TO DRAW.AND.JUMP
REPEAT 4 [JUMP]
END
```

The forward number 42.43 in the procedure JUMP (above) is the length of the hypotenuse of a right triangle whose sides are of length 30. The length is computed using the Pythagorean Theorem: $c^2 = a^2 + b^2$, or c (the hypotenuse) = the square root of $a^2 + b^2$. In Logo, you can find the length of the hypotenuse by typing:

```
PRINT SQRT (30 * 30 + 30 * 30)
```

Other Challenges

You can probably come up with additional ways of drawing the tic-tac-toe design. Why not challenge your students to write one or more REPEAT statements to draw the design?

Think of how you would transfer other pencil doodlings to Logo. Sometimes, as in the case of the star, the pencil approach may be quite satisfactory. For other designs, you may have to use a non-traditional approach. It is fun to try to draw the same design in as many ways as you can!

How would you draw letters of the alphabet using the turtle? I'll bet that you draw them differently with the turtle than with a pencil! Try drawing some letters exactly as you would with a pencil. For example, for the letter H you would first draw two vertical lines (picking the pen up in between), then connect them with the horizontal bar.

It becomes quite interesting to compare pencil techniques with turtle techniques, doesn't it?

A former education and computer consultant, Dorothy Fitch has been the Director of Product Development at Terrapin since 1987. She can be reached at:

Terrapin Software, Inc.
400 Riverside Street
Portland, ME 04103

Cooperative Creations

Third Grade Dynamic Poetry
by Jandy Bird

Since LogoWriter provides an excellent opportunity for cooperative learning, we decided to design a project for third graders in the computer lab that utilized the "jigsaw" technique of cooperative learning in a LogoWriter project. (See Slavin, 1988, Maring and others, 1985.) The jigsaw technique is quite simple in concept, but it takes planning to organize it. The idea of jigsaw is that a group is given a project to do cooperatively. Then the group is temporarily split up and each member of the group gets training in different specific skills necessary for completion of the project—a piece of the puzzle. Each member becomes an "expert" in something, and it is each expert's job to return to the group and teach the skill to the other group members. The project we chose was the translation of a poem into a piece of dynamic writing using LogoWriter. Time did not permit the students to write their own poems for this project, so the poem selected was "Shapes"by Shel Silverstein, a favorite poet of most of the students.

The first step was to demonstrate dynamic writing so the students could understand what the project was about. The beginning of a simple story was shown to the students, simply printed on the page.

> Once upon a time there was a cat who lived by himself in a house at the edge of the forest. One day, the cat looked out of the window and saw the most beautiful white rabbit in the yard.

(The story used the cat, rabbit, house and tree because the shapes were already available and familiar to the students).

Then the students were shown a dynamic version of the beginning of the story. After seeing the dynamic version, they were asked to figure out how the procedures worked. The flip side of the dynamic story page was analyzed and discussed.

The procedures for the dynamic version of the story are as follows:

First, to draw the forest :

```
to forest
circler 60
pu
right 90
forward 20
setc 2
setsh 23
pd
shade
ht
end
```

This procedure uses the tool procedure, circler, for drawing a circle.

Then the house

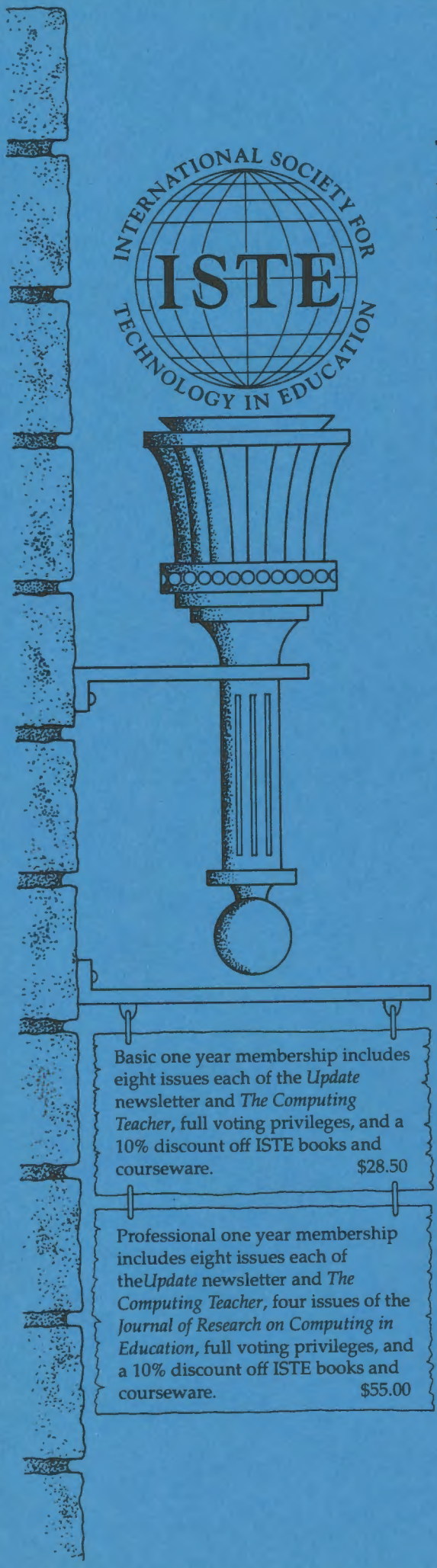
```
to house
pu
setpos [-45 25]
setc 5
setsh 20
st
pd
stamp
end
```

Then to bring in the cat

```
to cat
ht
setsh 21
setc 4
seth 180
pu
forward 20
st
repeat 10[forward 3]
end
```

Finally, to put the first sentence of the story together

```
to sen1
ct
rg
ht
print [Once upon a time there was a
   cat who lived by himself in a
   house at the edge of the forest.]
wait 60
forest
house
wait 20
cat
end
```



The ***International Society for Technology in Education*** touches all corners of the world. As the largest international non-profit professional organization serving computer using educators, we are dedicated to the improvement of education through the use and integration of technology.

Drawing from the resources of committed professionals worldwide, ISTE provides information that is always up-to-date, compelling, and relevant to your educational responsibilities.

Periodicals, books and courseware, *Special Interest Groups*, *Independent Study* courses, professional committees, and the Private Sector Council all strive to help enhance the quality of information you receive.

Rely on ISTE support:

- *The Computing Teacher* draws on active and creative K-12 educators to provide feature articles and carefully selected columns.
- The *Update* newsletter reaches members with information on the activities of ISTE and its affiliates.
- The *Journal of Research on Computing in Education* comes out with articles on original research project descriptions and evaluations, the state of the art, and theoretical essays that define and extend the field of educational computing.
- Books and courseware enhance teaching materials for K-12 and higher education.
- Professional Committees develop and monitor policy statements on software use, ethics, preview centers, and legislative action.
- The Private Sector Council promotes cooperation between educational technology professionals, manufacturers, publishers, and other private sector organizations.

It's a big world, but with the joint efforts of educators like yourself, ISTE brings it closer. Be a part of the international sharing of educational ideas and technology. Join ISTE.

Join today, and discover how ISTE puts you in touch with the world.

Basic one year membership includes eight issues each of the *Update* newsletter and *The Computing Teacher*, full voting privileges, and a 10% discount off ISTE books and courseware. $28.50

Professional one year membership includes eight issues each of the*Update* newsletter and *The Computing Teacher*, four issues of the *Journal of Research on Computing in Education*, full voting privileges, and a 10% discount off ISTE books and courseware. $55.00

ISTE, University of Oregon,
1787 Agate St., Eugene, OR 97403-9905.
ph. 503/346-4414