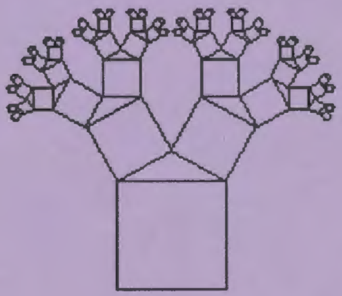
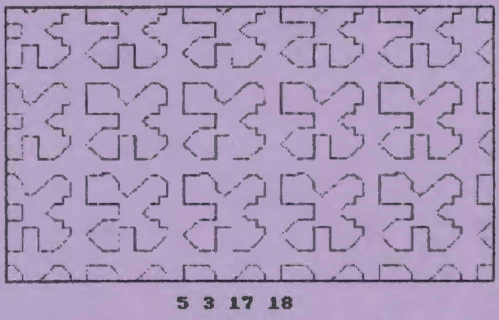
Journal
of the ISTE
Special Interest Group
for Logo-Using
Educators

LOGO EXCHANGE

Spring 1992 Volume 10 Number 3





International Society for Technology in Education

LOGO EXCHANGE

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

Founding Editor
Tom Lough

Editor-In-Chief
Sharon Yoder

Associate Editor
Judy Kull

Assistant Editor
Ron Renchler

International Editor
Dennis Harper

Contributing Editors
Eadie Adamson
Gina Bull
Glen Bull
Doug Clements
Sandy Dawson
Dorothy Fitch
Judi Harris
Mark Horney

SIGLogo Board of Directors
Lora Friedman, President
Bev and Lee Cunningham, Secretary/Treasurer

Publisher
International Society for Technology in Education
Dave Moursund, Executive Director
C. Dianne Martin, Associate Executive Director
Anita Best, Managing Editor
Talbot Bielefeldt, Associate Editor
Lynda Ferguson, Director of Advertising Services
Ian Byington, Production Assistance

LOGO EXCHANGE

Volume 10 Number 3 Journal of the ISTE Special Interest Group for Logo-Using Educators Spring 1992

Contents

Advertising space in *Logo Exchange* is limited. Please contact the Advertising Coordinator for space availability and details.

Logo Exchange (ISSN 0888-6970) is published four times a year by the International Society for Technology in Education (ISTE), 1787 Agate Street, Eugene, OR 97403-1923, USA; 503/346-4414. Subscription rates: Membership dues for SIGLogo are $20.00. $10 of this amount is for the subscription to *Logo Exchange* for one year. This publication was produced using Aldus *PageMaker*®.

POSTMASTER: Send address changes to *Logo Exchange*, ISTE, 1787 Agate St., Eugene, OR 97403-1923. Second-class postage paid at Eugene OR. USPS #000-554.

ISTE is a nonprofit organization with its main offices housed at the University of Oregon.

ISTE Order Desk: 800/336-5191
InterNet: ISTE@OREGON.UOREGON.EDU

Individual ISTE Membership: $46.00
Dues support the development, coordination, and delivery of ISTE services, including 8 issues of the *ISTE Update* newsletter, either 8 issues of *The Computing Teacher*, 4 issues of *Educational IRM Quarterly*, or 4 issues of the *Journal of Research on Computing in Education*, full voting privileges, and a 10% discount on ISTE books and courseware. Add $10 for mailing outside the USA.

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.

LX solicits articles on all topics of interest to Logo-using educators. Contact ISTE for submission guidelines. Opinions expressed in this publication are those of the authors and do not necessarily represent or reflect the official policy of ISTE.

It's *Beyond* Me!

by Sharon Yoder

Have you ever found yourself completely mystified by some Logo programming problem? Perhaps you asked a colleague or the local "expert" for assistance. Has your expert ever casually created a sophisticated solution to your problem that you *never* would have thought of? If you are like me, then your first question is always "How did you *ever* think of that?" Perhaps your expert replied, "Oh, I learned that in a Pascal class" or "That's a common programming problem—I learned how to do that in a data structures class."

Indeed, many, if not most, people using Logo have had little or no computer science training. Logo is most often taught in educational environments, while languages like Pascal tend to be taught in computer science environments. Logo classes focus on pedagogical issues, on educational philosophy, and on learning styles. On the other hand, Pascal classes focus on such issues as programming style, data structures, and algorithm analysis. The result is that even a teacher well trained in the use of Logo in the classroom is ill prepared to deal with complex programming and/or computer science issues.

Generalists and Specialists

I have seen a parallel phenomenon in my other teaching areas. For example, I recall teaching keyboarding to seventh graders in the early days of using computers in the classroom. One student disagreed with my rules for placing fingers on the home row of keys. I assumed the student simply misremembered the instructions his elementary school teacher had given him. Not so—several students who had been in the same class agreed with his criticism. As the day passed, I encountered a number of students, all from the same elementary class, with the same misinformation. At first I was very annoyed that this teacher had taught her students incorrectly, but the more I thought about the problem, the more I realized I had little reason to be upset. Keyboards were very new in the elementary classroom. We had not provided inservice on keyboarding to all of our teachers, and yet we were expecting them to teach it. There was certainly no prerequisite of expertise in teaching typing for getting an elementary teaching certificate in my state. And, after all, it actually seems very reasonable to shift your right hand one key to the left since, in the correct position, your right little finger is almost always over some punctuation mark.

Another striking example occurred when I was teaching my advanced seventh grade math students why division by zero is not allowed and what it meant to say that the result of such a division would approach infinity. A number of the students provided "alternate" explanations that had come from their elementary teachers. As a mathematician by training, I was quite annoyed. Again, however, some reflection made me realize that I came to understand division by zero well into my mathematics major in college. Why should I expect elementary teachers to have that same knowledge?

Problem Solving With Technology

Last fall I taught a class in computer applications. Students in this class chose a major piece of applications software they wanted to learn, e.g., *Microsoft Word, ClarisWorks, PageMaker*, and so forth. During class time, we shared successes, failures, and problems. We spent a lot of time thinking about solving problems in a technology environment. Often my role was to provide technical explanations, such as where and how fonts are stored, what kinds of graphics formats are available, what problems are involved in writing fast sorting algorithms, how a network handles transfer of data, and so forth. In other words, I often brought to the class my computer science knowledge upon which they could then build their conceptual models.

As computers become increasingly easy to use and increasingly prevalent in schools, we will encounter more and more minimally trained users who have no training in computer science. While teachers will increasingly encounter problems with technology, there is little reason that we should expect them to be knowledgeable about computer science concepts when there is so much else they need to know. Classroom teachers often have outside experts on whom they can rely for answers to technical questions in other fields, such as art, music, or math. Clearly we also need to have experts who understand computer science, hardware, and software at a level that they can assist teachers in problem solving.

The Teacher Training Problem ... Again

However, herein lies the problem. Increasingly as we train technology education leaders, we are moving away from the teaching of any computer science. Ten years ago, educators wanting to work in the area of technology took courses in computer science departments. But that has become increasingly difficult. Most computer science departments have a rather high level of prerequisites for many of these courses, often calculus and/or discrete mathematics. At the same time, colleges of education are moving away from teaching or requiring much programming for masters or doctoral level students who want to become leaders in the field of technology in education. The result is an ever-increasing number of technology education "experts" who do not have enough foundation in the computer science field.

While we might not expect the classroom teacher to be an expert in touch typing, we certainly expect the typing teacher to know correct hand placement, and much more. While we might not expect the elementary classroom teacher to know the reasons why division by zero is not possible, we certainly would expect the math coordinator or high school upper level math teacher to be able to explain the reasons to novices. Why, then, do we feel it is acceptable for computer specialists in schools to have little or no knowledge of computer science?

This is certainly not a problem with an easy solution, and I'm certainly not going to solve it by writing one editorial.

However, I can make several suggestions for the Logo programmers among you. First, find someone knowledgeable about computer science who is willing to answer questions. You might find a computer science teacher or someone at a local college or university. Then when you have a Logo problem—or other computer problem—ask that person for help and *ask for an explanation*. Gradually you will learn enough to solve more of the problems for yourself.

Second, do some sophisticated Logo reading. Take a look at the MIT Press Logo series. You probably want to start with Brian Harvey's *Computer Science Logo Style, Volume 1*. This is *not* an easy book. After a number of years of working with it, I *still* pick up tidbits. Brian's book, and some of the others in the series, give you real insights into Logo from a computer science point of view.

Even Your Editor Encounters Problems!

And where do *I* go to seek help? Well, if the problem appears to be with the version of Logo I am working with, I contact the company that makes the version of Logo: Terrapin, LCSI, and Paradigm all know who I am! If it's a programming/computer science problem, I first check some of my books—often the *Computer Science Logo Style* series—or I send electronic mail to Brian Harvey.

Oh ... but now I must run. My electronic mail program just beeped—maybe it's the answer to my latest question from Brian!

Note: The MIT Logo series of books is now available through ISTE. Call or write for a price list.

Sharon Yoder
SIGLogo
1787 Agate Street
Eugene, OR 97405
BITNET: Yoder@Oregon
InterNet: Yoder@Oregon.uoregon.edu
Ph: 503-346-2190

HOUSE Revisited

by Tom Lough

Quarterly Quantum

Welcome to the QQ. For new readers, this column (which may never be sufficiently brief!) offers a quantum (a fixed quantity or amount) of information, news, commentary, or what-have-you each quarter.

Every so often, I return to my hometown of Elkton, Virginia, nestled in the beautiful Shenandoah Valley. It is small as towns go; only about 1,600 people live there.

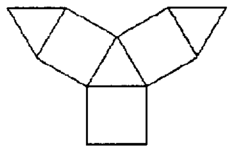
One of my favorite activities in Elkton is to visit the houses in which my family lived while I was growing up. After spending their first couple years of married life in a small apartment, my parents rented a charming little bungalow next to what was then the local fairgrounds. What an exciting location for a young lad!

After I completed the third grade, we moved to a farmhouse out in the country. This was a difficult change at first, but I soon grew to love farm life, and spent many happy hours on the banks of the Shenandoah River. Just before I started ninth grade, we moved back to town into a rental house for one year, before my parents finally were able to purchase our first house a couple of blocks away.

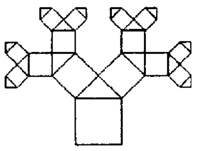
As I drive past these houses, memories wash over me. Such a sentimental journey satisfies some deep need for revisitation, for taking one more look at something associated with a previous personal experience. I would expect that many readers have felt similar stirrings.

There is another house I revisit from time to time. It is simply called HOUSE, or sometimes TO HOUSE. Yes, it is the familiar TRIANGLE and SQUARE combination of *Mindstorms* fame. Every year or so, I succumb to an urge to play HOUSE once again. I delight in the memory of the decade-old "awakening" this particular procedure provided me.

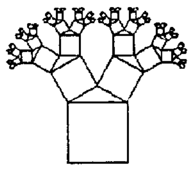
My most recent HOUSE call was unexpected. I was reading through a book on fractals when I spotted an illustration that inspired a project based on HOUSE. The idea was to draw a HOUSE and use the two top roof lines of the TRIANGLE procedure as bases for two additional HOUSEs of the same size. This gave me a Y-shaped figure. Then came the hard part—how to repeat that pattern indefinitely.



Later, I got the crazy idea that the pattern would be more interesting if the two top roof lines of the TRIANGLE were shorter than the base. This would produce two smaller HOUSEs on top of the first HOUSE. I took the easy way out by changing TRIANGLE to draw a right triangle with the 90-degree angle at the top and two 45-degree angles at the bottom. This produced a surprisingly familiar pattern when repeated—a binary tree:



Finally, I got the totally crazy idea of changing TRIANGLE so I could draw the pattern with any angle at the base. After a bit of work (understatement), I figured it out. I felt good. I learned a lot.

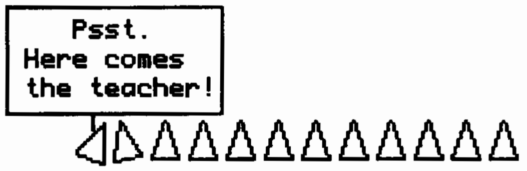


Dear readers, I am not telling you this tale to suggest that you also try to figure out this pattern. But I am suggesting that there are many reasons to return to the HOUSE procedure from time to time. You just might be surprised at what you discover about the old home place. I'd love to hear about your journeys.

FD 100!

PS: I would be happy to mail a copy of a procedure listing for this column to any reader who sends me a stamped self-addressed envelope.

Tom Lough
Founding Editor
PO Box 394
Simsbury, CT 06070



Introducing... Multiple Turtles

by Dorothy Fitch

Beginner's Corner

What Are Multiple Turtles All About?

Your Logo turtle is probably a close friend by now. It listens when you talk to it. You enjoy spending time together. Perhaps it would even be jealous if it thought it had to share you with some of its friends—the other turtles that may be lurking somewhere in your language. In this column we'll meet these multiple turtles and learn to make friends with them, too!

Different versions of Logo offer different numbers of turtles. Early versions of Logo have a single turtle, *LogoWriter* has four turtles, and Terrapin Logo for the Macintosh has an unlimited number of turtles. Although Logo PLUS has just one turtle, it does have a STAMP command, which allows you to simulate multiple turtles. (We'll have to investigate that idea another time.)

People usually begin learning Logo by using just one turtle. That's fine. There is enough to think about with a single turtle. You need to become familiar with how to move and turn it, how its pen works, and how to get it to go where you want! But once you are in control, why not see what you can do with multiple turtles?

Multiple turtles are totally independent of each other. Each one has its own position and heading, it can wear its own unique turtle shape, its pen can be up or down, and it can be hidden or showing. It can even have its own pen color, pen pattern, or pen size, depending on the commands that are available.

There are actually only a few multiple turtle commands to learn: TELL, ASK, WHO, EACH, and TURTLES. (If your version doesn't have a TURTLES or ALL command, don't worry—you can write one that does the same job. See the Notes section later in this column.)

A Lesson in Multiple Turtles

Many people, particularly young learners, first learn how the turtle moves and turns by experiencing it firsthand—by playing turtle. This lesson shows how you can introduce multiple turtles by playing "Multiple Turtle." Those who may feel self-conscious playing "turtle" may not feel quite as silly playing "multiple turtles." This experience will help them understand the value of playing turtle for people of all ages, no matter how many turtles are involved.

You can use the following dialogue with any group of students or adults who are ready to learn about multiple turtles. Each individual in the room plays the role of a turtle. Even if you have more people than your language has turtles, this is still a good way to show both visually and dramatically how the commands work. There's nothing like putting yourself in the position of the turtle, and it is especially effective when learning about multiple turtles, where things seem to become a little more complex. After going through these activities, you'll probably find that your audience can go to the computer and begin to use the new commands right away.

Preparation

Materials you will need:
- some blank index cards
- a marker to write on the cards
- an overhead projector
- some overhead transparencies
- a marker to write on the transparencies.

1. Count out as many blank index cards as you have people in the room (not including you).
2. Number the cards sequentially beginning with 0.
3. Give each person in the room a numbered card.

The Dialogue

You speak the lines marked YOU to the audience (feel free to use your own words). Write on a transparency the lines marked WRITE (or use a transparency you have already prepared with the text). The comments in parentheses may help you prompt your audience to respond correctly.

YOU: "What are the names of all the turtles I can play with today? I can find out by typing..."

WRITE:

```
PRINT TURTLES
```

(or `PRINT ALL` in some versions; see the Notes section).

YOU: "What do you think Logo will answer?" (Wait a moment to see if your "turtles" can guess the answer.)

WRITE:
```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```
(Go only as high as the numbered cards.)

YOU: "That's how many turtles I have altogether—all you turtles in this room. The command TURTLES tells me who is available to play with. But *who* will actually follow the commands I give? It may not be all of you!"

WRITE:
```
PRINT WHO
0
```

YOU: "OK. I'm talking to turtle 0. Who has that number? Let's try a command and see what happens."

WRITE:
```
Raise your hand.
```
(Only the person with card 0 should raise his or her hand. You may need to say to turtle 0, "Keep your hand up. I didn't tell you to put it down, did I? Pay attention!" This is spoken in fun, but reminds them all to follow directions explicitly, just as the turtle does.)

YOU: "Let's give a command to a different set of turtles. I can use TELL to get one turtle or a group of turtles to listen to me and follow my directions."

WRITE:
```
TELL TURTLES
```
(or `TELL ALL`)
PRINT WHO

YOU: "Who is listening to me now? What will PRINT WHO tell me?"

WRITE:
```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```
(as high as your turtles are numbered)
```
Raise your hand.
```
(Everyone should now raise his or her hand.)

YOU: "Great! Everyone's hand is up. You can put them down now. Let's play a game! We'll see how well you can follow directions."

WRITE:
```
TELL 0
Raise your hand.
Put your hand down.
TELL [0 1 2 3]
Raise your hand.
Put your hand down.
TELL TURTLES
Raise your hand.
Put your hand down.
TELL EVEN.NUMBERED.TURTLES
Raise your hand.
TELL ODD.NUMBERED.TURTLES
Raise your hand.
TELL TURTLES
Put your hand down.
```

YOU: "Does this remind you of another game?" (They will probably say, "Simon Says." If they don't, you can suggest it.)

"Let's add another command—ASK. This command lets you temporarily wake up any turtle or group of turtles, even if they aren't in the WHO list. They'll only listen to one instruction and then go back to sleep. Here's an example:"

WRITE:
```
TELL
[0 1 2 3]
Wave to me.
Stop waving.
ASK 4 [Wave to me. Stop waving.]
Wave to me.
```
(Turtles [0 1 2 3] should wave again because they are the turtles in the WHO list. Turtle 4 went back to sleep as soon as it stopped waving.)

YOU: "Who should be waving? Let's check."

WRITE:
```
PRINT WHO
0 1 2 3
```

YOU: "When would you want to use ASK? Here's an example to think about."

WRITE:
```
TELL READING.GROUP
```
Voice from the intercom: "Please send someone to the office with your attendance."
```
ASK JOHNNY [Please take the attendance
        to the office.]
        Read Page 4.
```

YOU: "Who should be reading page 4, the reading group or Johnny?"

(The reading group.)

"ASK is really a TELL command that is only active for one instruction line."

"Here is something you could type into Logo. Can you figure out what this instruction will do?"

WRITE:

```
TELL 2 [SETHEADING ASK 3 [HEADING]]
```

YOU: "It tells turtle 2 to ask turtle 3 what its heading is, then it sets its own heading to be the same. Turtle 2 sets its heading to wherever turtle 3 is pointing. When we're done with that instruction, we are talking to 2, not 3. That is a little complicated, but shows you a bit of the power of ASK. "

"One more new command: EACH tells Logo to ASK each active turtle to run a list of instructions. When it's each turtle's turn, it acts as if it were the only turtle in the WHO list."

WRITE:

```
TELL TURTLES
EACH [Say WHO you are (your
    turtle number).]
```

(Each person should say aloud his or her turtle number, starting with turtle 0.
You should hear "0", "1", "2", "3", "4", etc., up to the highest numbered turtle.)

```
EACH [REPEAT WHO [Clap]]
```

(Each person should clap as many times as their turtle number, starting with turtle 0, who doesn't clap at all! It's as if the following commands were given:

```
ASK 0 [REPEAT 0 [CLAP]]
ASK 1 [REPEAT 1 [CLAP]]
ASK 2 [REPEAT 2 [CLAP]]
ASK 3 [REPEAT 3 [CLAP]],
    and   so on...)
```

```
EACH [Say which month you were born
          in.]
```

(Each person, starting with 0, should give his or her birth month

YOU: "Here's an interesting example using EACH. See if you can figure out what to do."

WRITE:

```
TELL [0 1 2 3]
Stand up.
EACH [SETHEADING WHO * 90]
```

(Turtle 0 should face the front of the room: 0°. Turtle 1 should do a right 90 to face 90° —its WHO number * 90. Turtle 2 should face the back of the room, doing SETHEADING 2 * 90, or 180°. Turtle 3 should face toward the left, having done a SETHEADING 3 * 90, or 270°.)

Pop Quiz!

1. Which command tells you which turtles are listening to you? (WHO)
2. Which command lets you talk to a sleeping turtle? (ASK)
3. Which command tells you all the turtles you have to play with? (TURTLES) or (ALL)
4. Which command lets you talk to your turtles one at a time? (EACH)
5. Which command lets you address a particular group of turtles? (TELL)

Notes

If your version of Logo doesn't have a TURTLES or ALL command, which reports a list of all available turtles, you can write one as follows:

```
TO TURTLES
OUTPUT [0 1 2 3]    (include as many numbers as
                     you have people)
END
```

One of the previous example included the words EVEN.NUMBERED.TURTLES and ODD.NUMBERED.TURTLES. You can create your own procedures called EVEN and ODD and actually type instructions like TELL EVEN and TELL ODD to talk to different groups of turtles:

```
TO EVEN
OUTPUT [0 2 4 6 8 10 12 14]
END

TO ODD
OUTPUT [1 3 5 7 9 11 13 15]
END
```

At the Computer

When your students get to the computer, they can try using all these commands. Have them invent instructions that use TELL, ASK, EACH, and WHO.

You may want to have them experiment to find the difference between these two commands (assuming that SQUARE has already been defined):

```
TELL [0 1 2 3]
SQUARE
EACH [SQUARE]
```

Which instruction causes the turtles to draw a square one at a time?

Which instruction causes the turtles to draw a square all at the same time?

Sometimes you want all the turtles to draw all together, in unison. At other times, you'll want them to take turns. Practice these commands until you have complete control over all your turtles.

Ideas for Using Multiple Turtles

Now that you know how to control multiple turtles, what can you do with them? Here are some ideas to get you started.

- Dress your turtles in the shapes of cars or horses, and have a race. Use the instruction EACH [FORWARD 10 + RANDOM 20] so that they will each go forward a random amount (between 10 and 29).

- Move your turtles to different spots on the screen and create a simple connect-the-dots game. Have each turtle label its position with its WHO number. Send the turtles home and have one of them draw a line from number to number.

- Give each turtle the shape of a letter and scramble words by placing the turtles at random positions on the screen. You can also stamp the shapes to form word puzzles.

- Give one turtle the shape of a locomotive and several others the shape of a train car. Line them up so that they look like a train. Use TELL to talk to all of them so that when you give a forward command, they all move together. You have just created a megashape!

- Draw a checkerboard by lining up eight turtles at the bottom of the screen. Have them each draw a column of squares, all at the same time.

Happy Logo adventures with multiple turtles!

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

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

CompuServe: 71760,366
Internet: 71760,366@COMPUSERVE.COM
207/878-8200

1991 ICPSC Logo Winners

by Donald T. Piele

The International Computer Problem Solving Contest (ICPSC) is an annual event that challenges teams throughout the world to create original solutions to a set of five problems within two hours using a programming language. The purpose of the ICPSC is to challenge the very best problem solvers in Grades 4-12, yet make it possible for beginners to have some success. Teams of one to three members each enter the contest in the Elementary (Grades 4-6), Junior (Grades 7-9), or Senior (Grades 10-12) division. This is a report of the Logo Division winners.

Elementary Logo Division

The best solutions to the Elementary Logo Division problems were written by a team of two girls, Chelsea Jones and Robin Featherstone, from St. Michaels University School in Victoria, B.C., Canada. Chelsea, an outstanding student in science, mathematics, and computers, spends a lot of time on her home computer. Last year she was a member of a team of three girls who also solved all five problems and ranked fourth internationally. Chelsea has also completed the training for the NASA student astronaut program. This was Robin's first try in the contest. In addition to working with Logo, she likes riding horses. The advisor for the team was Dr. Lex McMaster.

Junior Logo Division

The team of John Sullivan and Mark James from Coláiste an Spioraid Naoimh, Bishopstown, Cork, Ireland, ranked number one in the Junior Logo Division. Both boys are 15 years old and have excellent mathematical ability. John was recently awarded a medal in the Canadian Mathematics Competition.

This was the fourth time John and Mark entered the ICPSC. They began in the Elementary Logo Division and have been ranked internationally each year. Mark placed second in the Junior Logo Division in 1989—the same year he was the Irish National Logo Champion—while John was the runner-up.

In his spare time Mark enjoys painting miniature figures and reading, while John likes tennis and cycling. The local contest director and team advisor was Michael D. Moynihan.

Senior Logo Division

The team of David Frink and Charles Powell from Enloe High School in Raleigh, North Carolina, placed first in the Senior Logo Division. David and Charles learned to love Logo in the third grade with their teacher Dorothy Hardin. They began competing in the ICPSC in the third grade and have continued up to this, their senior, year. By the fourth grade they were winning the local contest in Wake County and by the sixth grade they were solving all five problems and being ranked internationally.

This spring both Charles and David were selected by their geometry teacher to represent Enloe High School at the regional contest for the State High School Mathematics Contest. They were successful in the regional contest, and David went on to place third in the Geometry Division of the state contest. Both boys attribute much of their success in geometry to their knowledge of Logo programming.

The Wake County contest director was Cathay Smith and the team advisor was Donna Frink.

How the ICPSC Works

The materials for running the contest along with a set of problems and sample solutions are distributed to local contest directors who have agreed to conduct the contest locally during the last week in April—usually the last Saturday. Local directors are free to offer local recognition by rewarding the best local teams in any way they choose. However, if a team solves all five problems, a difficult task, their solutions are sent to us for regrading and ranking among all teams that solved five problems. Certificates are sent to each team member in this category, and the top-ranked teams receive a plaque for their school.

The Local Contest

The person responsible for computer education in a local school system often coordinates a local ICPSC. A notable example is Cathay Smith, Computer Education Program Specialist for the Wake County Public School System. For the last nine years Cathay has organized a county–wide contest that serves as a model on how to run a local contest. This year, 67 teams from Wake County entered in the Open Division and 18 teams competed in the Logo Division. Many teams from Wake County solved all five problems, and three teams were ranked first in their division.

A set of procedures she uses to conduct the contest in Wake County is available upon request for teachers or program specialists.

1992 Contest

In 1991, the Logo Division problems were developed with the assistance of James King, a member of the Mathematics Department at the University of Washington. In 1992, we will continue with three Logo divisions: Elementary, Junior, and Senior.

New in 1992 will be a HyperTalk Division for students who have learned how to program using *HyperCard* for the Macintosh. The design of the contest will differ slightly, and programs will be judged by submitting *HyperCard* stacks on disk rather than program listings and sample runs on paper. This contest will be developed with the help of Joseph Hofmeister of Cincinnati Country Day School. Joe is the coauthor of *Learning With HyperCard*, published by South-Western Publishing Company.

The 12th Annual ICPSC will be held on Saturday, April 25, 1992, with Friday, April 24, and Monday, April 27, as the alternate dates. To learn more about the 1992 ICPSC, please send a request for a free copy of *ComputeIt!* This publication contains information about the 1992 ICPSC along with a registration form and an order form for previous sets of contest problems and solutions. Solutions are now provided in the language of the corresponding division.

Donald T. Piele
P.O. Box 085664
Racine, WI 53408

ICPSC: Brian Harvey Responds

As a results of a series of email interactions between your editor and Brian Harvey, the following letter was "intercepted" on its way to *The Computing Teacher*. In it, Brian refers to the following problems from the non-Logo ICPSC contest.

Elementary Division: Write a program that will create the letter E of any odd size N < 20. Test your program for N = 7, 9.

Junior Division: Write a program that will create the letter V of any even size N ≤ 20. Test your program of N = 9, 10.

Senior Division: Write a program that will create the letter A for any even size N ≤ 20. Test your program for N = 9, 10.

Since this issue of *LX* contains last year's ICPSC Logo contest results, Brian's thoughts seem appropriate for *LX* readers to consider. With Brian's permission, his letter follows.

Dear Editor:

Every year Donald Piele publishes his contest results in *The Computing Teacher*, and every year I have the same reaction: Why is Logo excluded from the "open" division and relegated to a separate contest of its own? This ghettoization helps perpetuate the myth that Logo is somehow a less serious language than BASIC, Pascal, or C, when in fact the opposite is much closer to the truth.

Perhaps the idea is that some people who've studied Logo have only studied graphics. If so, the divisions should be renamed "Graphics Division" and "Text Division"; programmers in those other languages should have the right to attempt graphics problems just as Logo programmers should have the right to attempt nongraphics ones.

In Logo we can, for example, solve the three letter-drawing problems from the article in the same way they'd be solved in any of the "open" languages, i.e., a FOR loop:

```
TO E :SIZE
STARS :SIZE
REPEAT (:SIZE-3)/2 [STARS 2]
STARS :SIZE-1
REPEAT (:SIZE-3)/2 [STARS 2]
STARS :SIZE
END

TO STARS :NUM
REPEAT :NUM [TYPE "*]
PRINT []
END

TO V :SIZE
FOR "I 1 :SIZE-1
   [SPACESTAR :I-1 2
   SPACESTAR 2*(:SIZE-(:I+1)) 1
   PRINT []]
SPACESTAR :SIZE-1 1
PRINT []
END

TO A :SIZE
SPACESTAR :SIZE-1 1
```

```
PRINT []
FOR "I 2 :SIZE/2
   [SPACESTAR :SIZE-:I 2
   SPACESTAR 2*(:I-2) 1
   PRINT []]
SPACESTAR (:SIZE/2)-1 :SIZE+1
PRINT []
FOR "I (:SIZE/2)+2 :SIZE
   [SPACESTAR :SIZE-:I 2
   SPACESTAR 2*(:I-2) 1
   PRINT []]
END

TO SPACESTAR :SPACENUM :STARNUM
REPEAT :SPACENUM [TYPE CHAR 32]
REPEAT :STARNUM [TYPE "*]
END

TO FOR :VAR :FROM :TO :STUFF
LOCAL :VAR
MAKE :VAR :FROM
IF :FROM > :TO [STOP]
RUN :STUFF
FOR :VAR :FROM+1 :TO :STUFF
END
```

Although Logo doesn't include a FOR loop as a primitive control structure, I wrote the one just given for use in this problem. I also found it helpful to write a procedure SPACESTAR that types a given number of spaces followed by a given number of asterisks.

To show off a little, I'll present a set of Logo tools that solve the same three letter-drawing problems in a more general way and that can easily be extended to other letter forms.

The general idea in each of these problems is that there is a pattern of certain numbers of spaces and asterisks; some of the numbers change by constant amounts between lines (e.g., the number of spaces inside the V decreases by 2 between lines); and certain lines are exceptions that don't follow the pattern. For example, consider the letter V. The general pattern is some number of spaces, then asterisks, then spaces, then asterisks. For an eight-line V, the numbers start at

```
[0 2 12 1]
```

(that is, 0 spaces, 2 asterisks, 12 spaces, 1 asterisk). The change between lines is

```
[1 0 -2 0]
```

(That is, we add 1 to the number of spaces on the left, add 0 to the number of asterisks on the left arm of the V, subtract 2 from the number of spaces inside the V, and add 0 to the number of asterisks on the right arm of the V.) The printing pattern is

```
[. * . *]
```

in which the periods represent spaces and the asterisks represent themselves. (I'm using periods to represent spaces both because it's easier to see in the program listing and because spaces are used in Logo to separate list elements, so it's messy, although possible, to have a space as a list element itself. The program will, however, actually print spaces in its output when we put periods in the pattern.) The only exception is the last line, in which the printing pattern should be

```
[. - - *]
```

In this pattern, the hyphen means not to print anything, not even a space, for the corresponding number.

For each letter shape, we have to specify an initial set of numbers, a set of change values, a standard pattern, and perhaps some exception patterns. Here's how we'll do it:

```
TO V :SIZE
LETTER :SIZE ;how many lines
   ;initial list of numbers:
   (LIST 0 2 2*(:SIZE-2) 1)
   ;change values for numbers:
   [1 0 -2 0]
   ;normal print pattern:
   [. * . *]
   ;exception list:
   [-1 [. - - *]]
END
```

In the exception list, a positive number is a line count starting from the top; a negative number is a line count starting from the bottom. So the number -1 in the V exception list refers to the bottom line. An additional possibility, seen in the next example, is that an equal sign instead of a number indicates an exception on the middle line.

The letter E has two stars on every line, except for the first and last lines, with SIZE stars, and the middle line, with SIZE-1 stars:

```
TO E :SIZE
;how many lines
LETTER :SIZE
   ;initial list of numbers:
   (LIST 2 :SIZE :SIZE-1)
   ;change values for numbers:
   [0 0 0]
   ;normal print pattern:
    [* - -]
    ;three exceptions:
    [1 [- * -] -1 [- * -] = [- - *]]
END
```

The letter A has two exception lines, one at the top and one in the middle:

```
TO A :SIZE
LETTER :SIZE ;how many lines
    ;initial list of numbers:
    (LIST :SIZE-1 2 -2 1)
    ;change values for numbers:
    [-1 0 2 0]
    ;normal print pattern:
    [. * . *]
    ;two exceptions:
    [1 [. - - *] = [. * * *]]
END
```

Okay, now we have to make this work by implementing the LETTER procedure. It uses a subprocedure, LINE, which has one extra input, the number of the line we're on:

```
TO LETTER :LINES :NUMBERS :CHANGES
    :DEFAULT :EXCEPTIONS
LINE 1 :LINES :NUMBERS :CHANGES
    :DEFAULT :EXCEPTIONS
END

TO LINE :LINENUM :TOGO :NUMBERS
    :CHANGES :DEFAULT :EXCEPTIONS
IF :TOGO = 0 [STOP]
DISPLAY :NUMBERS (SELECT :LINENUM
    :TOGO :DEFAULT :EXCEPTIONS)
PRINT []
LINE (:LINENUM+1) (:TOGO-1) (ADDVEC
    :NUMBERS :CHANGES) :CHANGES
:DEFAULT :EXCEPTIONS
END
```

DISPLAY is the procedure that types asterisks and spaces on a single line. Its two inputs are the list of numbers for this line and the print pattern that applies to this line. That print pattern is chosen by SELECT, which outputs the default pattern unless one of the exceptions is relevant to the current line. Finally, ADDVEC performs a vector addition, i.e., it takes two lists of numbers and adds the corresponding elements of the two lists. That's how we apply the changes to the numbers as we move to the next line:

```
TO DISPLAY :NUMBERS :PATTERN
IF EMPTYP :NUMBERS [STOP]
REPEAT FIRST :NUMBERS
[DISPLAYONE FIRST :PATTERN]
DISPLAY BUTFIRST :NUMBERS
BUTFIRST :PATTERN
END

TO DISPLAYONE :CHAR
;do nothing for - :
IF EQUALP :CHAR "- [STOP]
;space instead of . :
IFELSE EQUALP :CHAR ".
[TYPE CHAR 32]
[TYPE :CHAR]
END

TO SELECT :LINENUM :TOGO :DEFAULT
    :EXCEPTIONS
;no exception applicable:
IF EMPTYP :EXCEPTIONS [OUTPUT
:DEFAULT]
;matched count from top:
IF EQUALP FIRST :EXCEPTIONS :LINENUM
  [OUTPUT FIRST BUTFIRST
:EXCEPTIONS]
;matched count from bottom:
IF EQUALP FIRST :EXCEPTIONS
(0-:TOGO)
    [OUTPUT FIRST BUTFIRST
:EXCEPTIONS]
IF AND (EQUALP FIRST :EXCEPTIONS "=)
    ;middle line for odd-size letter:
    (OR (EQUALP :LINENUM :TOGO)
    ;middle line for even-size
letter:
    (EQUALP :LINENUM :TOGO+1))
    [OUTPUT FIRST BUTFIRST
:EXCEPTIONS]
OUTPUT SELECT :LINENUM :TOGO
:DEFAULT (BUTFIRST BUTFIRST
:EXCEPTIONS)
END

TO ADDVEC :A :B
IF EMPTYP :A [OUTPUT []]
OUTPUT FPUT (SUM FIRST :A FIRST :B)
    (ADDVEC BUTFIRST :A BUTFIRST :B)
END
```

Of course this solution is more complicated than my earlier, BASIC-like version. But this flashy technique allows me to show off Logo's ability to deal with variable-size data aggregates, in this case the lists of numbers that control the patterns of the lines. (By the way, these procedures work for both even and odd sizes of any letter, with no limit other than that imposed by the width of the screen.)

I haven't included the part about prompting the user to enter a number; instead, I've made use of Logo's provision of procedures that take inputs. You'd draw an eight-line letter A by typing

```
A 8
```

If you prefer the prompt-and-read style, we can do that too:

```
TO DRAWA
TYPE [Enter an even number:]
A (FIRST READLIST)
END
```

My program is longer than a solution that is written from scratch for each letter shape. But now that I've written the program, I can handle other letters without having to rewrite it all:

```
TO H :SIZE
LETTER :SIZE
   (LIST 1 :SIZE-2 1)
   [0 0 0]
   [* . *]
   [= [* * *]]
END

TO J :SIZE
LETTER :SIZE
   (LIST (:SIZE-1)/2 1 (:SIZE-1)/2)
   [0 0 0]
   [. * .]
   [1 [* * *] -1 [* * .]]
END

TO N :SIZE
LETTER :SIZE
   (LIST 1 -1 1 :SIZE-2 1)
   [0 1 0 -1 0]
   [* . * . *]
   [1 [* - - . *] -1 [* . - - *]]
END
```

The LETTER procedure does not work for letters like B and D, in which the change per line isn't constant; nor does it work for R, in which two entirely different patterns are used in the top and bottom halves. But the approach could be generalized, with some extra design effort, to encompass those cases. The big idea illuminated by this approach is that of data-directed programming; pattern lists like [. * . *] customize the program so that the instructions can remain unmodified no matter what letter we want to display.

I couldn't resist throwing in a computer science lesson, but let me end by recalling my original purpose in writing this letter. I appeal to the organizers of the contest to stop discriminating against Logo, and to teachers of computer science to consider Logo as a powerful vehicle for advanced general-purpose programming.

Sincerely yours,

Brian Harvey
Computer Science Division
University of California
Berkeley, CA 94720
bh@cs.Berkeley.EDU

Logoed Mottos

by Judi Harris

Can Logo be used to solve "real-world" problems, or must Logo challenges be contrived and solved from within simulated contexts, such as turtle graphics or LEGO/Logo? Recently, I received an interesting request by electronic mail that showed me that Logo can, in certain situations, function as *more* than an educational tool.

The Challenge

The electronic message was from Talbot Bielefeldt, the patient and witty associate editor for the International Society for Technology in Education (ISTE). In part, he asked,

> Would you be interested in a programming problem for dear old ISTE? I am part of a committee charged with coming up with a new tagline or motto for the organization. The current one is "Educational Leadership in the Age of Information Technology." Quite a mouthful. It's so cumbersome, we never use it.
>
> After an initial brainstorming session, we came up with a list of concepts, which I expanded [upon] with the aid of an electronic...thesaurus. Our next step is to see how these might be constructed into some mottos. It seems to me there are several syntax patterns appropriate for a tagline. They include:
>
> participle/noun
> ([example:] "Making Waves")
> participle/preposition/noun
> ([example:] "Working for Change")
> preposition/adjective/noun
> ([example:] "For Better Education")
> noun/participle/noun
> ([example:] "Teachers Helping Students")
> preposition/verb/noun
> ([example:] "To Change Education")
>
> All essentially complete the sentence "ISTE is..."

At this point, like any Logo fan(atic), I was smiling and already planning how I might procrastinate meeting my less interesting deadlines so that I could begin work on this special Logo challenge.

Jumping In

Talbot had included five brainstormed word lists (nouns, verbs, adjectives, prepositions, and participles) at the end of his message. I downloaded the message, copied the word collections to a new *LogoWriter* page, and turned each into a procedure that OUTPUTs a list that contains all words in a particular category. These procedures are listed below.

```
TO NOUN
OUTPUT [education change future
   world technology solutions oppor-
   tunity challenge vision questions
   issues earth society humanity
   outlook prospect resolution an-
   swer key innovation transforma-
   tion difference guidance knowl-
   edge learning teachers learners
   students teaching advocate voca-
   tion passion force power light
   liberation renewal creation des-
   tiny individual center] END

TO VERB
OUTPUT [guide know learn expect
   imagine envision transfigure
   change transform resolve answer
   solve serve summon dare question
   confront help motivate kindle
   encourage inspire awaken excite
   spark advocate light ignite lib-
   erate creating]
END

TO ADJECTIVE
OUTPUT [educational effective dy-
   namic meaningful true relevant
   instructional inspiring produc-
   tive current strong constructive
   practical useful active expert
   exciting rightful authentic cer-
   tain positive destined central
   far farthest]
END

TO PARTICIPLE
OUTPUT [guiding knowing learning
   expecting imagining envisioning
```

```
      transfiguring changing transform-
      ing resolving answering solving
      serving summoning daring ques-
      tioning confronting helping
      striving]
   END

   TO PREPOSITION
   OUTPUT [of by for to in into within
      inside [out of] with]
   END
```

That was the easy part. I paused to marvel at how fascination with language structures can translate into Logo programming opportunities as easily as can fascination with geometric figures or the physics of movement.

Choosing a Method

In another part of his message, Talbot had asked me,

> ...do you have some code lying around that you could modify to generate a heap of sample taglines? The program would use the syntax elements as variables and the word lists as input.

Not bad for an only-occasional Logo tinkerer, eh? Talbot had recognized a "Logo-like" problem and mentally constructed a tentative blueprint for the procedures that could give the committee their desired output. He went on to predict,

> Many of the combinations would be garbage. A few might be quite amusing. But it would be one way [for us] to get a start...

At this point, I realized that Talbot was requesting output from a set of Logo procedures that would generate possible *permutations* of words according to each of the five syntactical structures mentioned earlier in his message. Those combinations could be generated one at a time and at random, or *all* possible permutations could be generated and listed.

Simple Sampling

Telling Logo to generate random combinations according to predetermined syntactical structures is an easy task. The PICK tool can be used to make random selections from within each of the five word type lists (nouns, verbs, adjectives, prepositions, and participles):

```
TO PICK :LIST
OUTPUT ITEM (1 + RANDOM COUNT :LIST)
   :LIST
END
```

Combined with a simple PPN procedure that concatenates random selections from named word lists, the PICK tool and the primitive PRINT quickly form possibilities for the ISTE motto in a participle/preposition/noun format:

```
TO PPN
OUTPUT (SENTENCE PICK PARTICIPLE
   PICK PREPOSITION PICK NOUN)
END
```

Typing PRINT PPN may cause the computer to produce

```
ENVISIONING WITH INNOVATION
```

or...

```
SUMMONING TO CHALLENGE
```

or...

```
GUIDING INTO LIBERATION
```

Procedures similar to PPN could be coded to yield potential ISTE mottos in other formats. For example, TO PVN could be used to generate random preposition/verb/noun permutations:

```
TO PVN
OUTPUT (SENTENCE PICK PREPOSITION
   PICK VERB PICK NOUN)
END
```

Of course, concatenation tools such as PVN could output

```
INTO RESOLVE EARTH
```

as easily as they could produce

```
TO GUIDE TRANSFORMATION
```

The disadvantage to giving tools such as these to Talbot and the motto committee is, of course, that someone would have to keep typing some version of the PRINT PICK (concatenation type) procedure execution until Logo randomly generated meaningful phrases that could be used as motto candidates. I suspected that the committee wanted a more time-efficient way to review motto ideas than using procedures that produced one option at a time.

Permutation Possibilities

The *real* challenge, I was soon to discover, was to write a set of Logo procedures that would generate and print all possible permutations of mottos within each format type. Since the numbers of motto options in the participle/preposition/noun structure alone were more than 7,700, I decided to create three smaller lists with which to experiment with Logo permutation.

Inspired by the November ice storm that rages outside my door as I write this column, and the knowledge that many of you will be enjoying spring weather as you read about Logo mottos, I decided to create lists of SEASONs, STATEs of being, and DESCRIPTORs:

```
TO SEASON
OUTPUT [WINTER FALL SUMMER SPRING]
END

TO STATE
OUTPUT [[WILL BE] WAS IS]
END

TO DESCRIPTOR
OUTPUT [COMING HERE]
END
```

There are 24 possible permutations of words in a season/state/descriptor format. They are:

```
SPRING IS HERE
SPRING WAS HERE
SPRING WILL BE HERE

SUMMER IS HERE
SUMMER WAS HERE
SUMMER WILL BE HERE

SPRING IS COMING
SPRING WAS COMING
SPRING WILL BE COMING

SUMMER IS COMING
SUMMER WAS COMING
SUMMER WILL BE COMING

FALL IS HERE
FALL WAS HERE
FALL WILL BE HERE

FALL IS HERE
FALL WAS HERE
FALL WILL BE HERE

WINTER IS COMING
WINTER WAS COMING
WINTER WILL BE COMING

WINTER IS COMING
WINTER WAS COMING
WINTER WILL BE COMING
```

As you can see, each element from each category must be combined with every other element from every other category. A multiple (embedded) recursive structure, with which as many calls to a COMBINE procedure as exist possible permutations, is one way to solve this challenge:

```
TO COMBINE :ITEM1 :ITEM2 :ITEM3
IF EMPTY? :ITEM1 [STOP]
COMBINE BUTFIRST :ITEM1 :ITEM2
   :ITEM3
IF EMPTY? :ITEM2 [STOP]
COMBINE :ITEM1 BUTFIRST :ITEM2
   :ITEM3
IF EMPTY? :ITEM3 [STOP]
COMBINE :ITEM1 :ITEM2 BUTFIRST
   :ITEM3
MAKE "NEW.COMBINATION (SENTENCE
   FIRST :ITEM1 FIRST :ITEM2 FIRST
   :ITEM3)
IF NOT MEMBER? :NEW.COMBINATION
   :COMBINATIONS [ADD.TO.LIST]
END

TO ADD.TO.LIST
MAKE "COMBINATIONS LPUT
   :NEW.COMBINATION :COMBINATIONS
END
```

As you can see in the last three lines of COMBINE, the computer checks to see if a newly formed combination of single elements from :ITEM1, :ITEM2, and :ITEM3 already exists in the "COMBINATIONS list before adding it with the ADD.TO.LIST procedure. This is a necessary step, since the embedded recursive-calls to COMBINE cause some duplicate concatenations to be formed.

The permutation process is initiated with a superprocedure that I called PERMUTE3:

```
TO PERMUTE3 :ITEM1 :ITEM2 :ITEM3
MAKE "COMBINATIONS [ ]
COMBINE :ITEM1 :ITEM2 :ITEM3
INSPECT :COMBINATIONS
END
```

PERMUTE3 initializes the value of the "combination collector" global variable, "COMBINATIONS, then directs COMBINE to execute, using the lists specified as values for the local variables :ITEM1, :ITEM2, and :ITEM3. Notice that all possible combinations are generated and stored before any are printed for the user to see. INSPECT (adapted from Bull and Cochran's tool with the same name) is the subprocedure that prints the combinations, organizing them into several columns to make viewing easier:

```
TO INSPECT :LIST
IF EMPTY? :LIST [STOP]
INSERT FIRST :LIST
IFELSE (FIRST CURSORPOS) < 50 [IN-
   SERT CHAR 9] [INSERT CHAR 13]
   INSPECT BUTFIRST :LIST
END
```

Please note that all procedures in this article are written in *LogoWriter*, but can be easily adapted to function in any full-featured version of Logo.

Time Is All We Need

Excited that the permutation procedures worked so well with my sample season/state/descriptor lists, I eagerly invoked the superprocedure with PARTICIPLE, PREPOSITION, and NOUN as inputs:

```
PERMUTE3 PARTICIPLE PREPOSITION NOUN
```

Forty-five minutes later, my Macintosh SE/30 with 5 megabytes of RAM was still permuting. It is fortunate that my Mac seems to have more patience than I do with repetitive tasks, since it will faithfully generate and store potential ISTE mottos for hours on end while I am involved in more interesting work.

When Sharon Yoder, the editor of the *Logo Exchange*, saw the code I had written for making permutations, she suspected, as I did, that there was a simpler, more efficient way to tell the computer to make more than 7,700 combinations. An electronic mail message to Brian Harvey, the "Logo Wizard" <grin>, was all that was needed to provide a much more elegant, economical solution to this interesting problem.

Expertise Is Ultimate Time-Efficiency!

Brian suggested that we use all of the word-list procedures and the INSPECT tool, along with these three procedures that form a new Logo operation:

```
TO COMBINE :LISTOFLISTS
IF EMPTY? :LISTOFLISTS
[OUTPUT [[ ] ]]
OUTPUT PREPENDS (FIRST :LISTOFLISTS)
   (COMBINE BUTFIRST :LISTOFLISTS)
END

TO PREPENDS :TOPS :BOTTOMS
IF EMPTY? :TOPS [OUTPUT [ ]]
OUTPUT SENTENCE (PREPEND FIRST :TOPS
   :BOTTOMS) (PREPENDS BUTFIRST
   :TOPS :BOTTOMS)
END

TO PREPEND :TOP :BOTTOMS
IF EMPTY? :BOTTOMS [OUTPUT [ ]]
OUTPUT FPUT (SENTENCE :TOP FIRST
   :BOTTOMS) (PREPEND :TOP BUTFIRST
   :BOTTOMS)
END
```

To generate, for example, all the possible motto combinations that contain participles, prepositions, and nouns, we would type:

```
INSPECT COMBINE (LIST PARTICIPLE
   PREPOSITION NOUN)
```

One of the most powerful aspects of Brian's solution to this problem is that COMBINE can be used to permute any number of elements into motto combinations, unlike PERMUTE, which must be altered depending upon the number of elements to concatenate. That means that Talbot and his committee can have all of the output they requested with just five requests:

```
INSPECT COMBINE LIST PARTICIPLE NOUN
INSPECT COMBINE (LIST PARTICIPLE
   PREPOSITION NOUN)
INSPECT COMBINE (LIST PREPOSITION
   ADJECTIVE NOUN)
INSPECT COMBINE (LIST NOUN
   PARTICIPLE NOUN)
INSPECT COMBINE (LIST PREPOSITION
   VERB NOUN)
```

As I write this column this evening in Omaha, I am envisioning Talbot and the motto committee's reactions as they receive five very *large* text files by electronic mail when they return to work after a (hopefully) restful weekend. Somewhere in the more than 30,000 possibilities that their word lists were used to create, there just *may* be a new ISTE motto waiting to be discovered and adopted. It delights me that Logo—and more importantly, people's creativity and teamwork—

fueled the search. Why not share a similar collaborative language adventure with *your* students?

Judi Harris works in the Department of Teacher Education at the University of Nebraska at Omaha as an assistant professor of educational technology. Her teaching, research, and service interests include Logo (of course), developmental sequencing in educational hypermedia materials design, telecomputing for K-12 teachers, and the restructuring of teacher education paradigms.

Judi Harris
514J Kayser Hall
Department of Teacher Education
University of Nebraska at Omaha
Omaha, NE 68182

BITNET: JHarris@unoma1
Internet: JHarris@Zeus.unomaha.edu

Oops!

In the December/January 1990/91 issue of *Logo Exchange*, there was an article entitled "Easy Map Drawing with *LogoWriter*." It included a listing of a program to use to draw a map. Well, drawing using that program certainly was *not* easy. The program contained a number of bugs. Following is a listing of the program that has been tested in Macintosh *LogoWriter* and therefore should run without any changes in any version of *LogoWriter*.

```
to map
name pos "startingpos
name heading "starthd
ht
pd
startdrawing [] 0
end

to startdrawing :list :counter
if :counter = 25 [copylist stop]
name readchar "x
if :x = "f [forward 5 (name lput
   [forward 5] :list "newlist)]
if :x = "r [right 15 (name lput
   [right 15] :list "newlist)]
if :x = "l [left 15 (name lput [left
   15] :list "newlist)]
if :x = "g [right 90 (name lput
   [right 90] :list "newlist)]
if :x = "t [left 90 (name lput [left
   90] :list "newlist)]
if :x = "d [if (first last :newlist)
   = "forward [pe back 5 pd name
   butlast butlast :newlist
   "newlist]]
startdrawing :newlist :counter + 1
end

to copylist
flip
bottom
print []
print [pu]
(print [setpos] (list :startingpos))
(print [seth] :starthd)
print [pd]
print.list :newlist
end

to print.list :commands
if empty? :commands [stop]
print first :commands
print.list butfirst :commands
end
```

Logo Foundation Announced

In the fall of 1991, Dr. Seymour Papert announced the formation of the Logo Foundation, a not-for-profit education organization devoted to the support of Logo-using educators throughout the world. Michael Tempel, formerly director of educational services with Logo Computer Systems, Inc., is president of the Foundation.

The Logo Foundation has been a dream for many years. It became a reality with commitments of initial founding from Logo Computer Systems Inc., of Montreal, and Logo Japan Inc., of Tokyo.

The creation of the Logo Foundation comes at a time of profound change in the ways in which computers are being used in schools. For many years, computers have been concentrated in "computer labs" with specialists providing instruction. Moving computers into regular classrooms could have a major impact on the learning culture.

"Logo is one of the most widely used and enduring educational computer environments. It is designed to support active, learner-centered education," said Tempel. "Yet there are many teachers who, while sharing Logo's vision of education, do not use Logo. They may see computers as irrelevant or even alien to their way of thinking and working. The Foundation will make a special effort to work with these educators."

The activities of the Logo Foundation will include:

- Comprehensive curriculum and staff development projects in a small number of schools and districts. As part of these projects, Foundation staff members and consultants will conduct workshops for teachers and assist them in using Logo in their classrooms. These projects will include the successful St. Paul Logo Project, which has served hundreds of teachers over the past nine years, and a new alternative school project being initiated in New York City's Community School District 3.

- A variety of workshops and seminars to be organized directly with schools and districts to meet their own specific educational needs.

- Organizational support for teachers wishing to start Logo User Groups, such as those existing in New York and Los Angeles. The Foundation will also establish communication links among these groups.

- Publication of informational brochures, research summaries, workshop materials, and curriculum units. These publications will include materials that emerge out of the Foundation's work in schools. Others will be contributed by Logo-using educators throughout the world. There will be a Foundation newsletter, to be published on a limited schedule during the 91-92 school year and monthly during the following school years.

- The establishment of a public software and materials library in the New York City Logo Foundation office.

For further information about the foundation contact

Michael Tempel, President
Logo Foundation
250 West 57th Street, Suite 2603
New York, NY 10107-2603
212/765-4918
FAX: 212/765-4789

Dots!!!

Eadie Adamson

Recently the chairman of the middle school math department at my school was thinking about asking his students to attempt to derive a circle. He wondered how the problem might be tackled with Logo. This set me to thinking. I wanted to try something other than the "classic" Logo circle. You know, the one that goes **forward 1 right 1**. I also remembered the procedures, embedded in some versions of Logo that drew a circle based on the radius. (See Judi Harris's LogoLinX column in the Fall 1991 *Logo Exchange* for these.) I wanted something else.

Is There a Dot?

For these explorations you will need to check whether the version of Logo you are using contains **dot** as a primitive. How can you find out? Just load Logo. Type the word **dot**. If Logo complains that it doesn't know how to "dot" or that there is no procedure named "dot," **dot** is not a primitive in your version. *LogoWriter* does not contain a dot, but it is an easy procedure to write. Here is a *LogoWriter* **dot**:

```
to dot
pd
forward 0
pu
end
```

If your version does not contain **dot**, you should check to see if **forward 0** produces a mark. Simply give the command **forward 0** and then hide the turtle. If you see no mark, change the dot procedure to read as follows:

```
to dot
pd
forward 1
back 1
pu
end
```

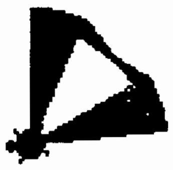
You need the turtle to move both forward and back so that its position after making the dot is unchanged. Without restoring position, you might get some strange results in the following explorations. (Actually, that might be interesting too, but not just now.)

Back to the Past

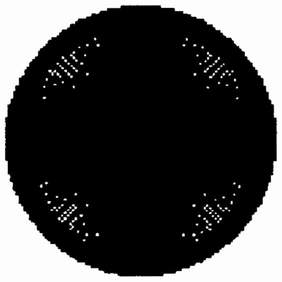
I remember when I was using a version of Terrapin Logo that did not have a fill command. I had several strategies for producing a filled circle. One of them was to draw a triangle, turn a little, and draw another triangle, repeating the process until the turtle returned to its starting heading. The procedure looked like this:

```
to fill.circle :size
repeat 360 [repeat 3 [forward :size
   right 120] right 1]
end
```

When this procedure is run, the circle begins to form:



This wasn't perfect, since it left some small spots unfilled:



Eventually I hit on the strategy of moving forward the distance of the radius, moving back the same distance, and then turning. Slow, but it generally did a better job:

```
to fill.circle :radius
repeat 360 [forward :radius
back :radius right 1]
end
```

This procedure was actually quite useful, for it led to a series of interesting projects. My filled circle could be adapted to make a half circle:

```
to halfcircle :radius
repeat 180 [forward :radius back
   :radius right 1]
end
```

and even a quarter circle:

```
to quarter.circle :radius
repeat 90 [forward :radius back
   :radius right 1]
end
```

At about the same time I discovered one of Ed Emberley's books, *Picture Pie.* My students and I followed up our filled circle work by trying our hand at creating Logo pictures a la *Picture Pie.* (If you don't know about Ed Emberley, you should! His picture books are a great resource for Logo users. You'll find them in most children's bookstores. I'll bet your school library has copies on the shelf too.)

The Circle Problem in a Different Vein

Recently at a New York Logo User Group meeting we scheduled a hands-on hour. Michael Tempel, president of the Logo Foundation, and I decided to pose the circle problem. After talking about the idea for a bit, we divided into small groups to explore alternative methods of constructing circles.

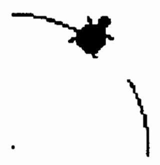
Thinking about this problem, I remembered the strategies I had used previously to draw a filled circle. There were strong similarities between these two problems. The difference from my earlier "fill" project was that this circle was not to be filled. Thus the problem, for the moment, was reduced to obtaining a result similar to my filled circle, but leaving only the outline of the circle. (I thought back to some other experiments with polygons—see "In the Spirit of Play: Playful Polygons" in the Logo Ideas column in the May, 1991, issue of *LX.*

Here Come the Dots

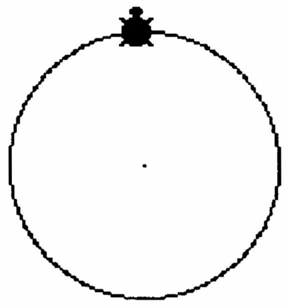
I proposed the "dot" strategy to the group. We wrote a procedure to trace the shape of a triangle, placing dots at the points of the triangle and leaving no lines between:

```
to tri :size
pu
repeat 3 [forward :size dot
   right 120]
end
```

We tried rotating the triangle and—voila!—our circle began to appear:



Yes, it did have a dot in the center. (If you think about why this occurs, it might give you a clue to what will happen in some of the following explorations.)



We could have eliminated the dot in the center; but instead, we pressed on to explore further. What would happen if we used a square rather than a triangle? We needed only to change the procedure name, the input to **repeat** and the angle to 90°. The result surprised us. It wouldn't be fair here to give you a picture of the result. Try it or think carefully about it and see if you can predict what we saw.

The odd result we got with a square prompted us to leave the circle problem behind. After all, in a way, we had solved it! Instead, we went off on a fascinating digression to explore the effects of rotating dotted polygons. We found ourselves attempting to predict what we would see, wondering about relationships between the number of sides or points of a polygon and the resulting picture. We ran out of time before we could address some other questions that arose:

- Is there a predictable relationship among the graphics produced by different dotted polygons?
- Which corner or vertex makes the outer circle?
- Which one makes the inner circle?
- What is the relationship between these points and the starting point?
- Is there a relationship among the diameters that can be computed? (We thought there would be.)

I leave the experiment for you and your students to try. You might make a chart of your results. Based on the chart, try to predict the results of using polygons with still more "sides." Can you deduce the relationship between the number of circles and the number of sides of the polygon? What might you say about the comparative areas of the circles?

Further Ideas

Here are some further thoughts. First you might write a generalized polygon procedure that puts dots at the apex of each side (you could pose this as a problem for your class to solve!):

```
to poly :sides :side
pu
repeat :sides [forward :side dot
   right 360 / :sides]
end
```

Then use the generalized polygon to define a new kind of circle procedure:

```
to cir :points :side
repeat 360 [poly :points :side
   right 1]
end
```

This will work fine for the first few experiments. Eventually you will find that turning *less* than 1 will help to make an outer circle in this multiple-circle process more likely to be a solid line:

```
to cir :points :side
repeat 720 [poly :points :side
   right .5]
end
```

The only problem is that it takes Logo so long to complete the drawing that it makes it come close to being a "Logo Overnight" experiment! Hiding the turtle while Logo draws will help some. You might, however, want to run the procedure on several computers at one time—and perhaps at a time when you could leave them running and move on to something else. (A lunch break, perhaps?)

To help you determine what you are seeing if you have color monitors, you might also have the dot for each corner of the polygon be a different color. This too is a nice programming challenge for students. It works best, of course, in versions of Logo with a wide range of colors.

Using colored dots, can you build into your procedure a counter that will tell you with, say, 50 rotations to the circle how many times the first dot out from the center encounters the second dot (if it does at all)? What kinds of dotted polygons form their circles by overlapping dots?

A Hands-On Follow-Up

Since many students may find even these "concrete" results difficult to believe, you might follow your computer session with a hands-on activity in the classroom. Use Pattern Blocks to perform the same experiment or have the students create sets of regular polygons with Logo, print them, and mount them on cardboard. Pin a corner of the shape to a sheet of paper and experiment with creating circles. Place a dot at each corner, then turn slightly and mark the corners again. Or make the shape into a minicompass by putting the pencil on one corner, and turning the shape while keeping the pencil at the corner. This will draw one circle. Repeat the process with the next corner. And again with the next. If this is done carefully, concentric circles will emerge.

A Different Circle

Here's another circular-problem idea for a "Weekend Logo" experiment: Write a procedure to draw a septagon, a seven-sided regular polygon:

```
to septagon :size
repeat 7 [forward :size
   right 51.4286]
end
```

Draw one septagon, then check the turtle's position by using

```
show pos
```

You will see that the turtle just misses returning home by a small fraction. One might tend to say that the fraction is so small it doesn't matter. But try drawing the septagon over and over without returning **home**. It will take a very long time before it becomes clear that a little error can be very important indeed. It just depends on the situation. For instance, if you're on a spaceship aiming for Uranus, that fraction would certainly cause you to miss your destination.

One of my students and I tried to find out what might happen with this errant septagon when the procedure runs for a long time. Our first experiment ran for nearly five hours on a Macintosh. This produced 100,000 septagons. We plan to set up a weekend experiment to see if what we think will happen over a much longer time is correct. First we'll estimate how many hours we want the experiment to run. Then we'll build a procedure to do the work. The number of hours divided by 5 will give us the approximate number of times to repeat drawing 99,999 septagons. While we're about it, we'll use the Macintosh *LogoWriter* clock primitive to report starting and ending times. Here's what we'll probably use:

```
to test
(print [Starting time:] clock)
repeat 11 [repeat 99999
   [septagon 50]]
(print [Finish time: ] clock)
end
```

What do you predict we will see on Monday morning? Would the result be different if the **septagon** procedure were written

```
to septagon :size
repeat 7 [forward :size right 360/7]
end
```

Explorations such as this can lead to discussions of "machine arithmetic." If this is not an area with which you are familiar, you might ask a computer science teacher from your school system to discuss this kind of error with your class.

References

Emberley, E. (1984). *Picture Pie*. Boston: Little, Brown and Company.

Eadie Adamson teaches at The Dalton School, where she works primarily with students in Grades 4-8, using *LogoWriter*, *LogoExpress*, and *LEGO TC logo*. She also works with math and language classes using *LogoWriter*.

Eadie Adamson
The Dalton School
108 East 89th Street
New York, NY 10128
212/722-5160

E-mail: LCSI LogoExpress BBS, New York and Montreal: EadieA
CompuServe: 73330,3266

Sanity and the Single Computer (Part 2)

by Judi Menken

The six-year old at the computer in the library corner has been working on a drawing of a boy playing ball. For 20 minutes she has been clearing and redrawing and is finally satisfied with the turn of the arm. The teacher now suddenly turns the lights off and announces clean-up time. Or the timer that gave her 20 minutes to work has sounded and the next child doesn't want to wait any longer. Or the 20 minutes of redrawing that arm was proving too frustrating, and what started out as an appealing challenge is now causing distress.

Up until a few months ago the above scenarios would have resulted in cries of "But I'm not finished!" My answer would have had to be, "You'll do it again tomorrow. Don't worry, your work always gets better with practice." The effects of this, in addition to the occasional tears, more frequent chair-slamming, and routine feelings of just making do, was that children would accept less than their best in order not to run out of time or patience.

I began a student disk to save the best work, but because it meant I had to interrupt the group I was working with, manually stop the Instant program the child was using, clear the startup section it contains, and get out the other disk, I only agreed to save the "best" work. This teacher-directed selection added a level of evaluation that again runs contrary to child-guided exploration and self-evaluation. This procedure also meant that I had to get involved again to reboot the child's work on demand, again encouraging cries of "Judi, I need ..." or "Get me ..." So, back to the late-night drawing board.

To deal with my problems, I developed a new Instant program, called **super.draw**. In this version the single-key command "X" brings up the **save** program. There is a **double.check** procedure so that the child can always get back into the drawing without clearing the screen. Many of the children are non- or beginning readers, so the steps have to be regular and the directions clear and simple. I numbered three double-sided disks from 1 to 6; four children's names were put on each side.

```
PUT YOUR DISK IN NOW.
NAME THIS PAGE.
```

directs the child to find his/her own disk from the box next to the computer. Many children had already been computer monitor for the month, so they knew how to handle a disk.

I was concerned that naming pages effectively and accurately would be a problem. But I found again that high motivation and focused procedures enable young children to achieve complex results. I might have to model naming a page the first time for a child, but I was amazed at how quickly they all picked up the idea of choosing a simple title for their work, one that would make it easy to recognize again.

These children do writing regularly, and invented spelling is an integrated component of many daily activities. If a child needs help spelling his/her page name, he/she knows by now to ask a nearby classmate.

My impetus for getting this **save** program organized was to disentangle myself even further from the children's independent use of my two classroom computers. I hadn't anticipated how much it would free them of the time constraints of classroom life. Children became willing to work for higher goals that they set for themselves; they became much more particular about what their work should look like. I found that what I had thought was just rambling around on the screen for many kids was actually a planned drawing, even if that plan was constantly being revised. Children were never at a loss for what to name their page.

Another unexpected bonus was what happens when the same child has to load his/her own disk. Any student planning to continue previous work has to learn Open Apple-S to stop the currently running Instant program, use the Escape key, and change the disks—no problem! I'm not even counting the reinforcement of working in a sequence and following directions, all those basic Logo extras for young children.

What I wasn't prepared for was the ease with which emergent readers could scan a lengthy contents list of very similar entries and zero in on "M.W.BOYS" rather than "S.M.2BOYS." How's that for a demonstration of the power of whole language? The seemingly nonsense letters have real meaning in that context. I had been withholding information and skills on the pretense that the children were too young, even though I am always preaching that primary-grade students can accommodate very complex sophisticated concepts and projects if these concepts and projects are arranged and presented in concrete, carefully defined building blocks.

Here I am writing about general curriculum. And I thought I was looking at just a Logo problem!

The Super.Draw Program

As the children gain experience the commands are also expanded to make more subtle moves and include more possibilities, including "X" to save their work.

```
to startup
ct
cg
pu
setpos [-100 50]
setc 2
pd
seth 0
stamp
pu
right 90
forward 20
label [Hi! I am the Turtle.]
setpos [-100 30]
label [Let's draw a picture.]
setpos [-100 10] ht
label [Type GO when you are ready.]
end

to doodle
cc
name readchar "key
if :key = "f [print "f forward]
if :key = "b [print "b back 10]
if :key = "l [print "l left 15]
if :key = "r [print "r right 15]
if :key = "e [pe]
if :key = "u [pu]
if :key = "d [pd]
if :key = 5 [setc 5 fill setc 1]
if :key = 4 [setc 4 fill setc 1]
```

etc.

```
if :key = "x [save]
if :key = "q [ct cg doodle]
if :key = "p [printout stopall]
doodle
end

to go
cg
ct
pd
seth 0
st
doodle
end
```

(For **printout** programs, see Part 1 of this article in the Winter 1991 issue of *Logo Exchange*.)

```
to save
cc
(type [Are you ready to SAVE this?]
   char 13)
type char 13
(type [Y or N?] char 32)
name readlistcc "okay
ifelse :okay = [n] [rerun]
   [save.this stop]
end

to rerun
cc
type [O.K. You can draw again.]
doodle
end

to save.this
cc
(type [Put your disk in now.]
char 13)
(type [Then name this page.]
char 13)
type char 13
make "answer first readlistcc
cleanup
np :answer
cc
(type [Put the other disk back in
   now.] char 13)
type [Then press RETURN.]
make "answer readchar
getpage "startup
```

Note: My disk is always too full for **getpage "startup**. If you have that problem too, use **leavepage** here instead.

```
end

to cleanup
if front? [flip]
top
search "to\ startup
if not found? [flip stop]
top
select
search "end
unselect
select
top
cut
top
search "to\ go
unselect
cf
select
```

```
    cd
    cd
    cd
    cut
    print "bottom
    flip
    end
```

This is the single-key program as my class uses it. The save program is written to match these commands. Both can be adjusted to whatever your preset limits are.

Thanks to Michael Tempel of LCSI for adding the select/cut procedures to automatically eliminate the startup parts of the drawing program. After hours of frustration alone, I finally followed the advice we give the kids and asked for help from someone who knew more. Thanks, Michael.

Judi Menken has taught primary-grade children for 14 years, the last 10 at a small, alternative public school in the middle of Manhattan. She uses Logo within an integrated curriculum classroom filled with guinea pigs, books, paint, pencils, and (oh yeah) computers. It *can* be done!

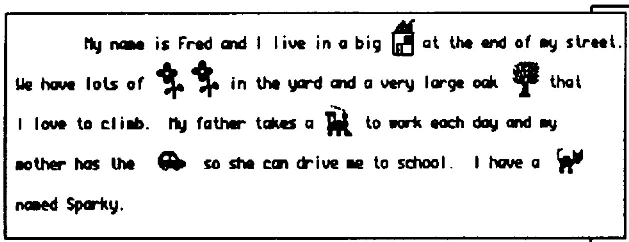
Judi Menken
Midtown West
328 W. 48th Street
New York, NY 10036
212/247-0208

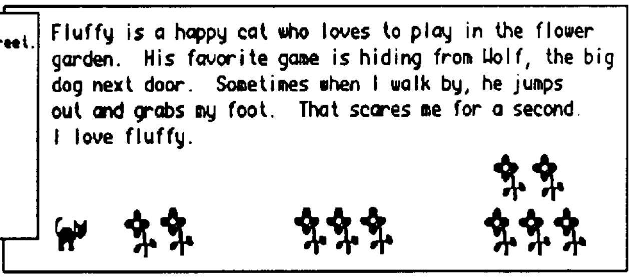
LogoWriter Activities for Readers

Build reading and writing skills in grades 1 through 6

- **Twenty-five scrapbook disks** with easy-to-follow student directions and teacher notes
- Supports **whole language, writing process,** and **traditional** approaches to instruction
- Works in conjunction with LogoWriter 2.0 or higher
- Apple IIe, MSDOS, and Macintosh versions
- Single classroom version - **$99.50**
- Building and district licenses based on population

Call for sample disks: 201 • 703-9676


My name is Fred and I live in a big at the end of my street. We have lots of in the yard and a very large oak that I love to climb. My father takes a to work each day and my mother has the so she can drive me to school. I have a named Sparky.


Fluffy is a happy cat who loves to play in the flower garden. His favorite game is hiding from Wolf, the big dog next door. Sometimes when I walk by, he jumps out and grabs my foot. That scares me for a second. I love fluffy.

Compuco, Inc. • 26-05 D Fair Lawn Avenue • Fair Lawn, N.J. 07410 • 201 703-9676

Hypermedia Links With Logo

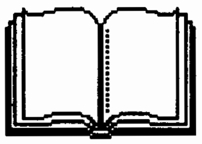
by Glen L. Bull and Gina L. Bull

The microworld, suggested by Seymour Papert, was an early and popular idea in Logo. A microworld is a small, self-contained universe that the learner can explore. The evolution of hypermedia has made it easier than ever to create such worlds. Two elements are required for hypermedia explorations:

1. a construct or metaphor, such as pages in a book or index cards in a stack of cards, that can be used as building blocks in the microworld's construction

2. some means of providing links among these elements.

LogoWriter uses the concept of pages in a book as its underlying metaphor, while *HyperCard* uses the concept of index cards in a stack of cards:



Metaphors for a Hypermedia Environment

The developer of a microworld might create a different room on each card in a stack or on each page of a book. In this type of microworld, hypermedia links provide a way to move from room to room.

Of course, the different kinds of universes that microworld developers can construct are as limitless as the imagination of the creators. A solar system could be created on the first page of a *LogoWriter* book, and subsequent pages could be devoted to each planet, with one page on Mars, one page on Jupiter, and so forth. When students develop such microworlds, they learn a great deal about the content which they are studying. The process of creating a page on Jupiter may require a trip to the library to find out more about this giant planet.

Hypermedia Buttons

Development of a hypermedia application requires some method for linking the elements of the world constructed. It should be possible to go directly to the *LogoWriter* page describing Jupiter without going through the pages on Mercury, Mars, Earth, etc. first. The **GetPage** command in *LogoWriter* provides a way of going to directly to any page in the book:

```
GetPage "Jupiter
```

Another hypermedia technique involves creation of buttons for linking elements together. For example, in *HyperCard* it is possible to create a button that, when pressed, takes the user to the page with Jupiter:

A useful variation of a visible button such as the one above is an invisible button that can be placed over an object of interest. For example, an invisible button placed over a picture of Mars might take the user to a screen containing information about the red planet.

Creating Buttons Within Logo

In the early 1980s an experimental version of Logo had the capacity for buttons, but thus far no commercially released version of Logo has had this feature. However, in the April, 1984, issue of *Logo Exchange* we described a technique that can be adapted to create hypermedia links in Logo similar to those produced with buttons in other applications.

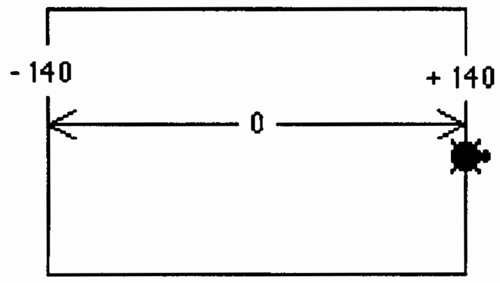
The method depends upon creating an invisible grid across the Logo screen. Each block in the grid is numbered; a Logo procedure called BLOCK? can tell us where the turtle is at any point in time. If the block the turtle occupies is over an object of interest, appropriate action may be taken just as though it were an invisible button.

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

The following setup procedure establishes the grid's basic coordinates. You will need to substitute the number of turtle steps across the width and height of your screen, depending upon whether you are using an Apple, IBM, or Macintosh computer. When we first developed the technique eight years ago, we were using an Apple II computer with Terrapin Logo. More recently we adapted the procedure for *LogoWriter* on Apple, IBM, and Macintosh computers as well.

```
TO SETUP
MAKE "SCREEN.WIDTH 280
MAKE "SCREEN.HEIGHT 180
MAKE "NO.OF.ROWS 10
MAKE "NO.OF.COLS 10
MAKE "BLOCK.WIDTH :SCREEN.WIDTH /
   :NO.OF.ROWS
MAKE "BLOCK.HEIGHT :SCREEN.HEIGHT /
   :NO.OF.COLS
MAKE "X.OFFSET (:SCREEN.WIDTH / 2)
   - 1
MAKE "Y.OFFSET (:SCREEN.HEIGHT / 2)
   - 1
END
```

The technique should work for any computer, but you will need to change the numbers for the variables SCREEN.WIDTH and SCREEN.HEIGHT so that they match the numbers needed by the computer you are using. If you are not sure of the number of pixels across your screen, you can move the turtle to the edge of the screen and print its location with the command PRINT XCOR. This will provide the turtle's X-coordinate. By doubling this number, you will have the width of the screen. You can use the YCOR command in a similar way to obtain the Y-coordinates that will provide the height of the screen.



The values we obtained in our measurements for the width (X-axis) and height (Y-axis) of the screens of different computers using *LogoWriter* were:

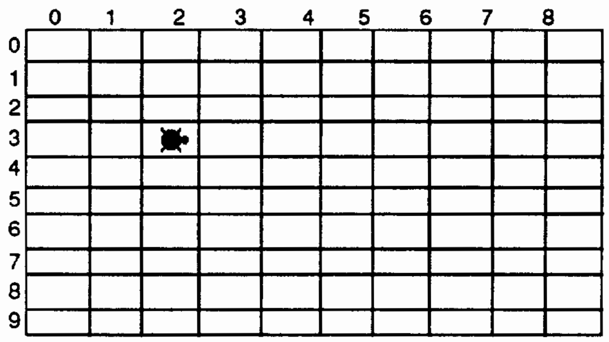
	Width	*Height*
Apple	280	180
IBM	320	190
Macintosh	495	221

After you have run the SETUP procedure, the BLOCK? procedure will tell which block the turtle is on at any given time. (Troubleshooting tip: be sure that there is no space between the colon and the variable name in variables such as :X and :Y; however, there should be a space *between* variables. Users who are using Version 1 of *LogoWriter* rather than Version 2 should consult the note at the end of this article.)

```
TO BLOCK?
MAKE "X INT (XCOR + :X.OFFSET) /
   :BLOCK.WIDTH
MAKE "Y INT (YCOR + :Y.OFFSET) /
   :BLOCK.HEIGHT
MAKE "Y (:NO.OF.ROWS - 1) - :Y
OUTPUT WORD :X :Y
END
```

For example, if the turtle is in the column labeled "2" and the row labeled "3", as shown below, the command PRINT BLOCK? would return a result of "23". Before running the BLOCK? procedure for the first time, it is necessary to run the SETUP procedure described above.

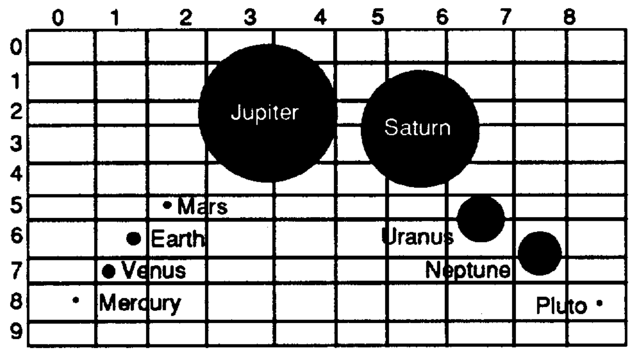
```
SETUP
PRINT BLOCK?
23
```



Each block can act as an invisible button. It will be necessary to name the buttons we are going to use. For example, on the following grid, Mercury lies within block 08, and Mars lies within block 25. Some objects may cover more than one block; for example, Neptune lies within blocks 86 and 87. In that case, both blocks should be named "Neptune" in the NAME.BUTTONS procedure:

```
TO NAME. BUTTONS
MAKE "08 "Mercury
MAKE "25 "Mars
MAKE "86 "Neptune
MAKE "87 "Neptune
END
```

There are a couple of ways to discover the block associated with an object. One method is simply to move the turtle to the object in question, and then type PRINT BLOCK?. The BLOCK? command will then print the number of the block the turtle (and the object) are occupying. We will also supply a DRAW.GRID procedure in the "Useful Tools" section that follows. This procedure can be used to draw an overlay of the grid on the Logo screen for planning purposes.



Each button must have an accompanying procedure that tells Logo what to do when the button is clicked. The procedure may simply print a message on the screen. If you are using *LogoWriter*, the procedure can also transfer the user to another screen. In the following example, clicking one of the buttons covering Neptune will take the user to another *LogoWriter* page with more information about that planet:

```
TO MERCURY
PRINT [Mercury is the planet closest
   to the sun.]
END

TO MARS
PRINT [Mars is known as the "red
   planet."]
END

TO NEPTUNE
GETPAGE "NEPTUNE
END
```

The CLICK.BUTTON procedure provides a way to click the buttons covering the objects on the Logo screen. The CLICK.BUTTON procedure checks to see if the block which the turtle is occupying has been given a name. If it has, the CLICK.BUTTON command runs the procedure associated with that block. For example, Block 25 was assigned the name of "Mars" in the NAME.BUTTONS procedure. Therefore, when the turtle has been moved to Mars and CLICK.BUTTON is typed, the MARS procedure will print "Mars is known as the 'red planet.'" It is important to observe the difference between brackets and parentheses when entering the CLICK.BUTTON procedure. In the following procedure, brackets are the outermost symbols with square corners, while the parentheses within the brackets have rounded corners:

```
TO CLICK.BUTTON
IF NAME? BLOCK? [RUN (LIST THING
   BLOCK?)]
END
```

You will need to run the NAME.BUTTONS procedure before trying the CLICK.BUTTON command for the first time. If you name more buttons associated other objects at a later date, it will be necessary to run the NAME.BUTTONS procedure again to activate these new buttons:

```
NAME.BUTTONS
CLICK.BUTTON
Mars is known as the "red planet."
```

Useful Tools

The numbered blocks covering the Logo screen are usually on an invisible grid. However, when you are designing a page it may be helpful to draw the actual grid lines on the screen. The DRAW.GRID procedure draws the grid lines. It uses two subprocedures, LINE and OVER. It is necessary to run the previously described SETUP procedure before running DRAW.GRID. (Troubleshooting Tip: Be sure there is no space between the minus sign and the number 2 in -2 below. If you are using *LogoWriter* Version 1 of rather than *LogoWriter* Version 2, you can omit the ROUND command in the DRAW.GRID procedure.)

```
TO DRAW.GRID
PU
HOME
OVER ROUND (:SCREEN.WIDTH / -2)
BACK :SCREEN.HEIGHT / -2
PD
REPEAT 10 [LINE :SCREEN.HEIGHT OVER
   :BLOCK.WIDTH]
FORWARD :SCREEN.HEIGHT
RIGHT 90
REPEAT 10 [LINE :SCREEN.WIDTH OVER
   :BLOCK.HEIGHT]
END

TO LINE :LENGTH
FORWARD :LENGTH
BACK :LENGTH
END
```

```
TO OVER :DISTANCE
PU
RIGHT 90
FORWARD :DISTANCE
PD
LEFT 90
END
```

The grid can be removed by setting the pen color of the turtle to 0 (white) instead of 1 (black), and running the DRAW.GRID procedure again:

```
TO REMOVE.GRID
SETC 0
DRAW.GRID
SETC 1
END
```

Enhancements

This column describes a way to create invisible buttons covering various objects on the Logo screen. When the turtle is moved to the object, either by using the traditional turtle commands (FD, RT, etc.) or by using the *LogoWriter* "Turtle Move" keys, the command CLICK.BUTTON activates the button. To use the "Turtle Move" keys, type Open Apple-9 in Apple *LogoWriter* or press Function Key 9—the key labeled F9—in IBM *LogoWriter* You can then use the Up Arrow and Down Arrow keys to move the turtle around the screen. Press the Escape key to return to the previous mode after the turtle has been moved to the desired position.

This basic framework can be enhanced by providing other ways to activate the CLICK.BUTTON command. The *LogoWriter* command WHEN (available in *LogoWriter* Version 2) provides one means of accomplishing this. The WHEN command programs a Control-key event. After the following line is entered, holding down the Control key and pressing the letter "x" will cause the CLICK.BUTTON procedure to be activated.

```
WHEN "x [CLICK.BUTTON]
```

This command will make it possible to move the turtle to any object on the screen covered by a button, and type Control-X to activate the button.

In the Macintosh version of *LogoWriter*, it is possible to use the Macintosh mouse to move the turtle around the screen. The Macintosh version of *LogoWriter* also provides a way to check whether the mouse button has been pressed (with the BUTTON? command). An additional enhancement, which we will leave as an exercise for readers with Macintoshes, would be to develop a way to activate Logo buttons with a click of the mouse button. Readers who have Apple or IBM computers with joysticks may wish to explore whether they can discover a way to activate Logo buttons with a click of the joystick buttons. (On those machines, the commands

```
Button? 0
```

and

```
Button? 1
```

report "True" or "False" depending on whether joystick buttons "0" or "1" are being depressed.)

It is important to run the SETUP and NAME.BUTTON procedures (which set up the invisible grid and name all of the Logo buttons that have been assigned in the grid) before running BLOCK? or CLICK.BUTTON. A STARTUP procedure will ensure that these procedures are run when the user first enters the *LogoWriter* page. If a Control-key event is defined so that Logo screen buttons can be activated with a Control-X, this command should be included in the STARTUP procedure as well:

```
TO STARTUP
SETUP
NAME.BUTTONS
WHEN "x [CLICK.BUTTON]
END
```

Summary

A number of hypermedia applications provide users with a means of controlling events through invisible buttons placed over objects on the screen. The SETUP, NAME.BUTTONS, and CLICK.BUTTON procedures provide Logo users with a method for creating similar hypermedia buttons within Logo.

When a button created in this manner is clicked, either by typing CLICK.BUTTON and pressing the Return key or by typing Control-X, the screen button at the location of the turtle is activated. The hypermedia event that occurs when a button is clicked depends upon the commands the developer places in a Logo procedure associated with the button. A message may be printed on the screen, a series of turtle commands may move the turtle to a different location on the screen, or the user may be transferred to another page.

The microworlds created with such hypermedia tools can be as varied as the creator's imagination. For example, one group of students visiting the geology corner of a science museum created a geology microworld. Screen buttons were linked to geologic samples on the Logo screen. Placing the turtle over a geologic sample and activating the button caused another screen with more information about the underlying mineral to appear.

A school directory is another project that could be developed through a hypermedia format. The floor plan of the school would appear on the first *LogoWriter* page. The turtle could be used to move from room to room. Clicking the underlying button in any room would take the user to another page with additional information about that area.

The underlying focus of any such project should be the knowledge the learner acquires in the process of creating the microworld. Educators who do not share the Logo philosophy and approach to learning might be tempted to spend hours creating a fully formed "GeoWorld" microworld for a class. In reality, a class is likely to learn more by going through the process of creating a microworld themselves and developing the knowledge that goes into the process than by using a template the teacher has created.

Additional Procedures for *LogoWriter* Version 1 Users

If you are using *LogoWriter* Version 1 rather than *LogoWriter* Version 2, you will also need to enter the following procedures since they are not built-in reporters (primitives) in Version 1:

```
TO INT :NUMBER
IF :NUMBER < 1 [OUTPUT 0]
OUTPUT FIRST :NUMBER
END

TO XCOR
OUTPUT FIRST POS
END

TO YCOR
OUTPUT LAST POS
END
```

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 system administrator in the Department of Computer Science 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
BITNET Addresses: GBull@Virginia, Gina@Virginia

...a conversation with Uri Leron....

by A. J. (Sandy) Dawson

Uri Leron dropped by to visit me in Montreal this summer as he was returning to Israel after spending another summer at the Institute for Mathematics and Computer Science Education (IFSMACSE) at Kent State University. I knew that this was the second year he had participated in this Institute so I was curious to find out exactly who the Institute was for, how it operated, and what made it unique. After all, when the likes of Uri Leron, Brian Harvey, Rina Zazkis, Al Cuoco, Phil Lewis, and Paul Goldenberg congregate for six weeks during the summer to do mathematics and computer science Logo style, it has got to imply that something significant is happening. The Institute was originally conceived of and organized by Ed Dubinsky of Purdue University, and Olaf Stackelberg of Kent State University, where the Institute is actually held. They began planning for the Institute in 1986. As described by Brian Harvey in a paper submitted to NECC '92, IFSMACSE was designed to foster *new sophistication in the curriculum, new methods of teaching, and new uses of technology* (Harvey, 1992).

The participants each summer are approximately 180 high school mathematics and computer science teachers, predominantly from Ohio and the surrounding states, half of whom attended the program the previous summer and half of whom are attending for the first time. Participants receive room and board plus about $300 per week. As Harvey (1992) describes it:

> They are divided into mathematics and computer science strands. During the first two years of IFSMACSE classes, there were two math strands, one focusing on new topics in mathematics, and the other on the use of computer technology in learning math. These were merged for the third year, starting in 1991, by combining courses from the two curricula.

The NSF-funded program just completed its third summer of operation. During the summer of 1991 Brian Harvey and Paul Goldenberg taught in the computer science strand, Rina Zazkis, Al Cuoco, and Phil Lewis taught in the computer and mathematics strand, and Uri Leron taught the second-year students in the pure mathematics strand. (The previous year he had taught the computer science strand with Brian Harvey.) Using the *ISETL* computer language,. Uri taught abstract algebra to these second-time students The key to what Uri saw as different about this way of teaching abstract algebra as compared to what he had done in the past, or what is normally done in university level mathematics courses, was in the way *ISETL* programming activities and small-group discussions were utilized to replace the traditional lecture method. Here is what he told me about his method (which he is developing jointly with Ed Dubinsky).

Uri: *ISETL*, which stands for Interactive *SET* Language, is not quite like other programming languages. It was invented and designed quite a few years ago by a well-known mathematician, J. T. Schwartz of the New York University's Courant Institute. (More precisely, Schwartz, designed an earlier version, called *SETL*, which was very powerful but ran only on mainframe computers and required compilation. Ed Dubinsky and Gary Levin designed the present, interactive version for microcomputers.) He did this not for any educational reasons. As a mathematician he wanted a computer language in which he could program more or less the way he thinks about mathematics. So his idea was to take the standard language of written mathematics and introduce the absolute minimal changes necessary to make it into something the computer could execute. It wasn't easy. In fact, it took him several years until he found someone who was able to implement it. So when you look at *ISETL* programs, it is almost like viewing any mathematics done with paper and pencil. If you are a mathematician, you can read and understand the program without knowing the programming language. One result of this is that to learn the language is almost like learning to write down mathematics. So students don't have to waste a lot of time learning computer *garbage*. Most of the things they have to learn are purely mathematical; very little time is spent on learning the language per se. Of the six weeks in Ohio, we spent only the first week on the language, but even during the first week, the students were already doing mathematics as they were learning the language. They were already writing down mathematical things like modular numbers and functions and the operations on these things. So even as they were learning the language, they were reviewing and in fact learning new mathematics.

Sandy: Would you describe one of these activities? I can't get a mental picture what one might look like.

Uri: Well, the activities normally would proceed from concrete to abstract and they were geared to teaching the important notions of abstract algebra, say, group theory. In the beginning we would start by writing the code, which is almost the same as a mathematical code, on the computer (in the *ISETL* language, of course), constructing some concrete examples of mathematical groups before actually talking about groups. All the time there is a general ongoing discussion. Students work in small groups, two or three or four in a group, and they are always supposed to talk. They actually talk about what they are doing, trying to guess how the computer will react to whatever was typed into it. The *rules of the game* prescribe that they always try to guess before they hit Return, writing down their guesses in their notebooks and then comparing this with the result from the computer. And actually this is the attitude we have: *If you guess correctly then you didn't learn anything new, so it was just a waste of your time; but if you didn't guess correctly then you are lucky, because you have something to learn from this activity.* So that's how we would work. Students start learning about the role of their classroom community, their own role in it, and the role of the computer. We want them to write mathematical code, but the difference is that when they write mathematical code in *ISETL* instead of with a pencil on paper, they can interact with—they can ask the computer to find what this identity element is, for example. The computer actually tells them whether or not there is one. And by playing with this code, the assumption is that the students not only make constructions on the computer, but that they also construct mathematical objects and processes in their mind. And this is really important: *For these mental constructions to happen, students should do the programming themselves rather than use any kind of prepackaged programs or software package.*

Sandy: Earlier you mentioned that you believed students didn't learn abstract algebra when they originally took it as undergraduate students because of the way they were taught: the lecture method, doing exercises, and things like that. But how are the exercises you have developed different from what a student might have done years ago? Then they listened to a lecture and were given a set of exercises to do. They did those exercises and checked the back of the book to see if they had the correct result. How is that any different from what you are doing with *ISETL*, where students type things into the computer, and if it is what the computer wants, that's fine, they go on into the next thing, and if it is not what the computer wants they go back and do something different?

Uri: Nasty question! It is different both because of the discussions that occur and the computer activities we have developed. To begin with, the students don't have the concept—they don't have the concept in their mind. They don't need to understand the stuff initially in order to do the computer activities; the computer activities can be done with partial understanding or with no understanding. And that understanding sort of emerges through the interaction with the activities using the computer. They start by typing something into the computer; in many cases they have no idea what the thing is supposed to be. In some cases they have a vague idea, but because this is a computer activity they can work with partial understanding. They can always *do* something. And by guessing what the results would be and comparing this with the computer the students gradually develop understanding. So each time they do an activity, they supposedly understand a little bit better the stuff they are working with, and all the time they are doing these activities, they are discussing it with their classmates. It really is quite different. You just have to go around and listen to these discussions. It is wonderful because they just talk about the kinds of notions you always wanted students to talk about but never really did. Here they are talking about this, more or less continuously arguing about these very notions. You can see that the students operate on a totally different level. It is wonderful when it comes out of them; it really shows you that they are thinking hard about what they are doing. I don't know how to elicit this kind of learning without the computer activities.

Sandy: So is one of the things you're after a different way of working in mathematics classrooms in university settings?

Uri: Yes, and we also think that this kind of work could go on in high school math classes. We don't yet have a lot of experience in high schools. All the work we have done so far was in university courses, but we're going to try it out very soon in high schools. And some of the successes were with college classes where the average talent was no higher than what you expect to find in high schools, so this makes us optimistic about what we can achieve at the high school level.

Sandy: I have a second nasty question! One of the things that was said about bringing computers into the teaching of mathematics is that we would in fact do mathematics in different ways, and the mathematics we created would be different. It seems to me from what you said that *ISETL*, in fact, doesn't do that. It's the same old mathematics. It is, in fact, directing students to do mathematics in the way it has always been done, using the terminology we have had for the last half century. But it doesn't allow for new mathematics to be created because it is training students to write and think about mathematics in the same way that has been around for years.

Uri: Isn't this the kind of question that Papert would describe as *technocentric*? I said *ISETL* is a tool, a tool that is appropriate and effective for dealing with some mathematical topics. It doesn't prescribe to you a particular way of doing mathematics or what kind of mathematics should be done. Take group theory, for example. It is a beautiful piece of mathematics, and we don't need to change that. But only very few students come to appreciate its beauty or make any intellectual use of it, and this is what we are trying to change. I think it is very important to distinguish between what's happening in the learning process, which is all about intuition and personal meaning and exploration, and the later stages when students come to connect with the general culture of mathematics, what I call the official mathematics. Now the meaning is personal, we try to reconstruct the whole meaning in our own ways. Part of the success of mathematics is that even though everybody constructs their own meanings, people can meaningfully discuss mathematics and more or less agree about what they are saying. Presumably you will have your own meaning of what I said, but we can still connect, communicate, and come to some kind of agreement about what we are talking about. This is a kind of meaning which is...

Sandy: ...social? Are you saying that there is a social aspect of what is acceptable, even within the mathematics community itself? You are suggesting, I gather, that you want the high school teachers in this program to go back to their high school students and teach that flavor of mathematics.

Uri: For me the most important thing is that the students will have a meaningful experience. Our hope is—and experience tends to support this, at least partially—that with these new methods of working with the computer, doing the construction of concepts with them, through the discussions, the students can, in a meaningful way, learn the mathematics. I think if we can manage to do this in a meaningful way, people wouldn't object to whatever mathematics is chosen for study, whatever is correct and interesting in the sense of the general culture of mathematics.

Sandy: Caleb Gattegno (1974) claims that only awareness is educable. Moreover, he said that awareness could be forced. What he meant by forcing awareness was that particular situations and activities, when presented to students, increase the likelihood that students are going to become aware of the mathematics being presented to them. Of course, there is no guarantee that this will happen. How does one know the likelihood has been increased? Experience tells you. You have done this before and you find that students get it if they do this, and they don't get it if they don't do this sort of thing.

Uri: This is exactly what I was trying to say.

On that note I had to drive Uri to the airport for his flight to Europe, and our conversation ended. However, during a break from editing the tapes of our conversation for this column, I spent a couple of hours talking with Benoît Côté at the Université de Québec 'a Montréal about his software package called *Les Deux Tortues* (Côté, sous presse). I was pleased to see that Benôit had put into his package, designed for upper elementary school-aged children and based on a Logo-like approach to the teaching of mathematics, many of the same things Uri was talking about for his class of high school teachers. Both Benôit and Uri want the learners with whom they work to *do* mathematics, to *discuss* mathematics, and to do so with the assistance of a computer. But neither wants the computer language to become the focus of the activity. A similar approach can be found with *Cabri-géom'Etre*, the geometry software package coming out of France (Baulac, Bellemain, & Laborde, 1989). In all three approaches, the focus is on learners exploring and mathematizing, and not on learning a computer language. They are tools that are appropriate and effective for dealing with some mathematical topics, as Uri noted above concerning *ISETL*.

As such I think all three approaches are on the right track.

References

Baxter, N., Dubinsky, E., & Levin, G. (1989). *Learning discrete mathematics with ISETL*. New York: Springer-Verlag.

Baulac, Y., Bellemain, F., & Laborde, J.-M. (1989). *Carbi-*

géom'etre: An interactive notebook to teach and learn geometry. Laboratory LSD2 (IMAG-UJF-CNRS), BP53X 38041, Grenoble, France.

Côté, B. (sous presse). *Les deux tortues: Un ensemble d'activités mathématiques pour le second cycle du primaire et le premier cycle du secondaire.* Montréal, QC: Machina Sapiens, 5780 Decelles Avenue, Montréal, QC, Canada.

Leron, U., & Dubinsky, E. (in press). *Learning abstract algebra with ISET*. New York: Springer-Verlag.

Gattegno, C. (1974). *The common sense of teaching mathematics.* New York: Educational Solutions Inc.

Harvey, B. (1992). *Beyond programming: A two-summer computer science institute for secondary teachers.* Manuscript submitted to the National Educational Computing Conference.

Uri Leron is an associate professor at the Israel Institute of Technology (Technion), Department of Science Education, Haifa 32000, Israel. He is the author of many articles about Logo and mathematics education, perhaps the two most well-known ones being "Logo Today: Vision and Reality," which appeared in 1985 in *The Computing Teacher* [12(5), 26-32], and "Structuring Mathematical Proofs," which appeared in the *American Mathematical Monthly* in 1983. Uri Leron's email address is: ttr0128@technion.bitnet.

Sandy Dawson has now returned to his duties as director of the Professional Development Program at Simon Fraser University in Vancouver, Canada. At the time of the preparation of this column, he was a guest of the Department of Mathematics and Statistics at Concordia University in Montreal, Canada. Sandy wishes to thank the department, and in particular Dr. Joel Hillel, for the hospitality shown him during his stay in Montreal.

A. J. (Sandy) Dawson
Faculty of Education
Simon Fraser University
Vancouver, BC
Canada V5A 1S6

Email address:
Userdaws@sfu.bitnet

Escher's Potato Stamps: A Microworld Programming Project

by Y. S. Give'on, Nitsa Movshovitz-Hadar, Rina Hadass

Escher's Game of Potato Stamps

In developing the activities described below, we were inspired by a specific game created in 1942 by M. C. Escher. The game was based on an ingenious use of stamps made of engraved potatoes. Escher's son, George, described a simplified version of this game as a way his father used to entertain him in long winter evenings.

Escher's idea was to make use of square potato stamps with a special structure. Their edges were divided into three equal parts by two points, and each one of these points was connected to another one by a line, as shown in Figure 1.

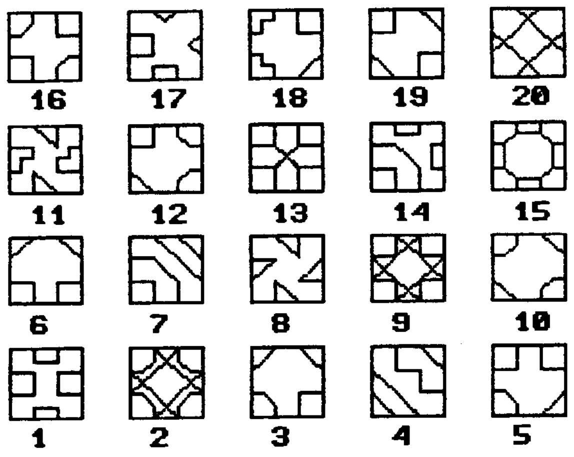


Figure 1. A sample of squared potato stamps made for Escher's design

The graphic examples were created with LCSI IBM PC Logo. The suggested procedures were written in that version of Logo, but they can be written, as is, in any other version, including *LogoWriter*, and for any other machine.

When any two such stamps are printed side by side, their lines connect through their joined end-points. Escher asked his son to form a square tile from four such stamps, as illustrated in Figure 2 and then to replicate this tile in a rectangular array to get a tessellation (Figure 3).

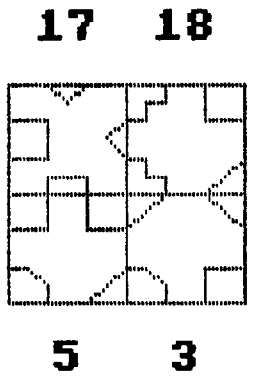


Figure 2. Four potato stamps attached to form a square tile

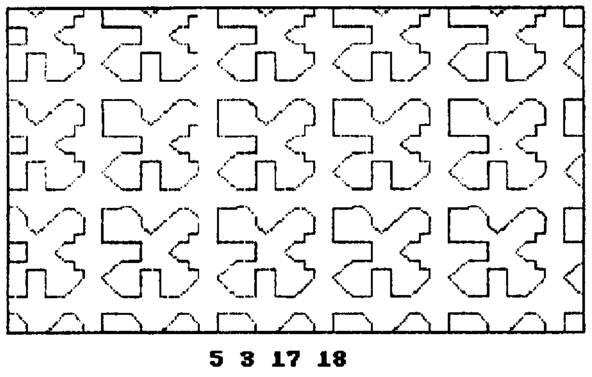


Figure 3. A tessellation made out of the tile in Figure 2

The number of different tessellations that can be produced in this fashion—using a small number of stamps to form tiles—is very large. In general, each basic stamp may produce different effects when rotated by 90 degrees, 180 degrees, 270 degrees, and 360 degrees, or when reflected horizontally, vertically, or by any of the two diagonals. For example, from 10 entirely asymmetric stamps, almost 10 million tessellations can be created.

The calculation of this count goes as follows. We consider only entirely asymmetric stamps. Therefore, each such stamp produces a set of 8 different stamps, by the four rotations and four reflections. Furthermore, we consider 10 stamps, none of which can be transformed into another by rotation or reflection. Therefore, these 10 stamps yield 80 different stamps. Now, in order to construct a 4-stamp tile, we choose any 4 of these 80 specific basic elements (80 choose 4 equals 1,581,580). We attach the 4 stamps together in 1 of (4!), or 24, possible tiles. Some, but not all of the 4! permutations of the 4 chosen stamps yield the same (infinite) tessellation. In fact, the set of 24 permutations of any such chosen stamps can be decomposed into 6 disjoint classes, each one containing 4 permutations that are equivalent in the sense that they yield identical infinite tessellations. Figure 4 shows one such class of four equivalent tiles made of the same 4 stamps. Thus, out of 10 totally different and asymmetric stamps, one may generate 1,581,580 x 24/4, i.e., 9,489,480, different tessellations.

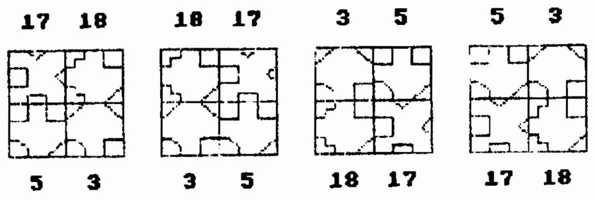


Figure 4. An example of four equivalent tiles

Fascinated by the richness of this game, we decided to simulate it by a computerized microworld. Logo seemed to be a natural language for such a project. It is presented here as a programming project for students, or for a teacher who may want to use the game as a ready-made educational environment.

The Programming Project

The proposed core program for Escher's potato-stamps game consists of two sets of procedures. The first set contains the specific procedures for the stamps themselves. The second contains the procedures for the production of the tessellation determined by any choice of a sequence of four stamps grouped into a single tile.

The composition of procedures for the first set, which produces stamps, is a worthwhile project by itself. It demands the rudiments of Logo programming only. As actual classroom experience indicates, this may become a very stimulating project, especially when the students are asked to create and evaluate their own designs for the stamps. In order to get there, some acquaintance with the effect of the 2 x 2 stamp tile on the tessellation is required. By playing with the tiling procedure, students will become gradually aware of the structural effects of the stamps in the tile on the overall design of the tessellation. Therefore, we suggest that the programming of the stamps be done in a special environment for hands-on, on-line experience. Such an environment can be prepared with a ready set of tiling procedures "buried" in it as tools.

The composition of the procedures of the second set is no less of a challenge. Let TILING be the procedure that draws the required tessellation according to an arbitrary choice of a sequence of four stamps. Since Logo is a rich environment for program design, one may compose TILING in many ways and in many levels of sophistication. Beginning Logo programmers who know only simple methods of programming would compose TILING in the following bottom-up manner: they would choose four formerly defined stamps, say STAMP1, STAMP2, STAMP3, and STAMP4. Then they would define NORTH, EAST, SOUTH, and WEST as the proper procedures for the appropriate "interfaces" between stamps. For example, if the size of each stamp is 30x30 turtle steps, the definitions of these interfaces can be very simple:

```
TO NORTH
PU
FORWARD 30
PD
END

TO EAST
PU
RIGHT 90
FORWARD 30
LEFT 90
PD
END

TO WEST
PU
RIGHT 90
BACK 30
LEFT 90
END

TO SOUTH
PU
BACK 30
PD
END
```

Finally, they would define TILE, say, for the production of the four-stamp tile, by:

```
TO TILE
STAMP1
NORTH
STAMP3
EAST
STAMP4
SOUTH
STAMP2
WEST
END
```

Now, TILING, the procedure that applies TILE repeatedly in several adjacent rows, can be defined in a very straightforward manner. Students who are aware of the practical value of total turtle trips (Abelson & diSessa, 1981) would program their stamps as total trips. Consequently, these interface procedures can be standardized, and TILE itself becomes a total trip as well. This way of programming is characteristic of beginners, because they do not possess the concept of high-level programming in which procedures can be used as inputs for other procedures. Each time they want to use other stamps, or another TILE, they must reprogram STAMP1, STAMP2, STAMP3, and STAMP4, or reprogram TILE, to suit their new choice.

Advanced students should be able to form TILE as a procedure with four variables accepting names of procedures of stamps. TILE should then execute the named procedures with the needed interfaces between them, as in the previous version of TILE. As experience in teaching Logo indicates, this idea is not foreign to beginning Logo students as well, and it may be suggested by such students if they are not stunted by limited programming languages.

The suitability of a programming language for educational purposes is measured, inter alia, by the ease in which bright ideas can be expressed with it. Logo allows one to define procedures (and functions) that can accept procedure names as data, and evaluate (i.e., execute) them precisely at any moment when required. This can be done in Logo by using any of the special verbs, such as RUN, REPEAT, IF, IFT, IFF, and CATCH. These verbs need lists for values of some of their arguments, and they execute the contents of these lists. This unique type of programming is called metalinguistic programming, or, in short, metaprogramming. It is available in Logo, as in all the dialects and progenies of LISP. There are many possible ways of defining TILE as a metaprocedure. In fact, this particular project is an excellent opportunity for bringing about the need for metaprogramming in a natural way (Zazkis & Leron, 1990). This is also a good opportunity for explaining the real meaning and operation of REPEAT in Logo as a metaprocedure, since it is not often taught and explained.

If the students are not versed in list processing, the teacher may offer them the following procedure "buried" in their environment:

```
TO EXECUTE :X
REPEAT 1 SENTENCE :X []
END
```

This is a very versatile procedure. It accepts words and lists and executes their contents. Students should familiarize themselves with its effect, and then they may use it in a better version of TILE. In this manner, with a proper change of TILING, they will have a general-purpose tiling procedure.

Advanced students can be asked to make the inventory of available stamps extensible by adding certain transformations that apply to any specific stamp. These may be the four rotations (by 90 degrees) and the four mirror reflections (by the square symmetry axes). The techniques of metaprogramming are very useful in the production of these transformations.

Students should become aware of the fact that Logo allows the modification of its primitives. In some versions of Logo this modification can be carried out directly by means of COPYDEF instructions. In some versions this can be done only after the instruction to MAKE the value of the special system variable REDEFP TRUE. For a start, they may use primitive modification in a metaprocedure that can operate on a given procedure to produce the mirror image of the outcome of the original procedure. The idea is to make RT borrow the definition of LEFT, and LT the definition of RIGHT, by means of COPYDEF, and then to EXECUTE the procedure for the original stamp, followed by COPYDEF instructions for the restoration of the original meanings of RT and LT. Next they will either attempt to program all the other reflections directly in a similar way, or they may discover that one mirror reflection can generate all the other reflections by its composition with rotations.

In one way or another, this programming project may function as a trigger for the concrete study of the group of these transformations of "the square," as well as of other geometrical figures. If this happens, the teacher can direct the students to use their procedures as a "transformation calculator," as a separate activity, in order to examine the outcome of various combinations of reflections and rotations.

The Game as a Microworld

The complete outcome of this programming project can be used as a microworld. Within this microworld, students can be asked to carry out two basic tasks.

1. They can select stamps and use proper commands to activate TILING and get a tessellation.

In this case, they can examine the effects of their selections and investigate various properties of the produced tessellation. For example, they may study the effects of figure and background as determined by certain features of the generating TILE. They may inquire as to the effect of symmetric stamps on the overall pattern of the tessellation. They also may raise the problem of how to characterize the identity of a tessellation and what can be the meaning of equivalence of tessellations. Actual experience with young students

(10 to 15 years old) shows that these problems come up naturally in such an activity. They may enjoy also the aesthetic quality of the tessellation.

2. They can be shown a completed tessellation produced by TILING.

The tessellation can be produced by a predetermined choice made by the teacher according to didactical considerations. For example, if the teacher plans to cover the subject of equivalent tessellations, the discussion can be based on a fixed set of four stamps and all their permutations. This may be achieved by a preprogrammed procedure that produces all the possible tessellations generated by a given set of four stamps. The tessellations also can be produced randomly by a procedure. The students then can be asked to decode the tessellation into its building blocks, i.e., any four stamps that could have produced it.

In order to make this microworld accessible to students of different levels of ability and experience, more features can be added to it. For example, if the previously described tasks seem to be too difficult, we may provide students with simple, introductory activities. For instance, we may have a feature that enables students to view the tessellation together with information about its structure. Figure 5 shows a suggestion for such an interface.

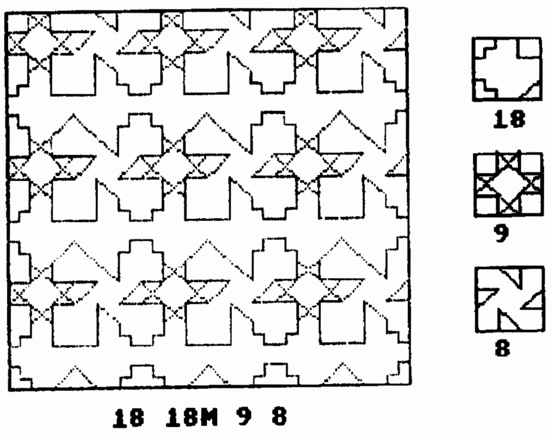


Figure 5. A tessellation and its basic stamps as assistance for decoding the generating tile

Additional help, such as a procedure that draws the grid of the stamps' borders, can be provided in order to aid the student's perception. In order to erase the grid lines, it is suggested that TILE is modified by including in it a SAVEPIC instruction so that the original tessellation is automatically saved as a picture file. This is done so that a procedure for the erasure of the grid may be defined as LOADPICing the saved original picture. Another helpful device is a procedure that draws an "empty tile," namely, a tile of four empty squares. This empty tile can be superimposed on the tessellation to show the actual tile used for the production of the tessellation. A more sophisticated feature would allow the students to change the position of the "empty tile" and move it around. This feature can help them search for several possible tiles that may manufacture a given tessellation.

On the other end of the difficulty-level range, the microworld should include features for the presentation of tessellations without these hints. One of those can be a procedure, say, RANDTILE, which randomly chooses a foursome of stamps out of a given basic set of stamps. Furthermore, we can make RANDTILE operate in WINDOW mode, locating the tessellation at a random POSITION. This feature prevents the outlines of the total tessellation from providing a hint about the chosen stamps. Figure 6 shows an example of a tessellation that is constructed and presented in such a manner. Decoding its generating basic stamps and tile is very difficult indeed.

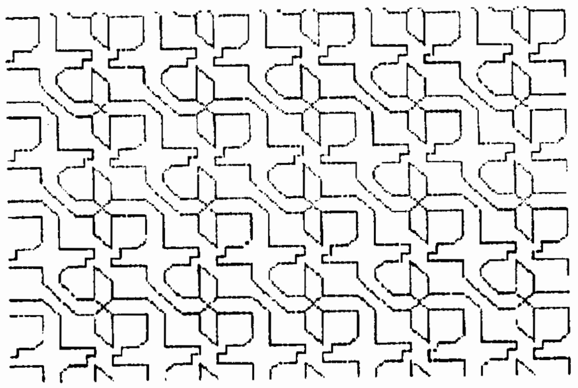


Figure 6. A tessellation with no boundary lines

Finally, one may add a stamp editor for creating new stamps and modifying existing stamps.

Concluding Remarks

Logo is an excellent flexible, multipurpose and authoring system for generating environments of investigative activities and constructionistic learning such as the environment suggested by Escher's Potato Stamp Game. Such environments, generated with Logo, are open to "on the spot" changes that can be made by the teacher who presents these microworlds to the students. Needless to say, this presupposes the teacher's mastery of Logo programming.

The microworld outlined in this article can be used as an educational tool in many types of activities. It can be used as a learning environment by itself, or combined with other tools in richer environments. For example, the stamps can be reproduced on rectangular pieces of cardboard that can be pasted on proper rectangular tokens to make up manipulatives. Students can become familiar with the essential ideas of the game by using these manipulatives before they attempt to operate the computerized procedures of the microworld

As our experience with a few students has shown, even the simplest activities within this microworld stimulate them to ask some very pertinent questions. For example, they spontaneously ask about the possible relationship that may hold between the properties of the stamps and the properties of the tiling design made of them. Under proper teacher guidance and intervention, such students are led to the understanding of some sophisticated ideas. The playful activities with this game of potato stamps are connected to some basic concepts of programming, to principles of computer usage, and, in particular, to the use of a computer-related environment in the study of complex structures. These ideas form the essence of computer science, both as an applied science and as an intellectual endeavor. We believe this microworld can provide a soft and aesthetic introduction to computer enlightenment as well as being a tool for mathematical education.

A version of this microworld is now being empirically investigated for its educational merits. In particular, we are interested in developing its potential use as an educational tool for low achievers. The preliminary data about the reactions of children to this microworld has motivated us in writing this article.

Recommendations for Further Reading

Details about Escher's work, and in particular about his idea of using potato stamps to create tessellations, can be found in Ernst (1976) and in Coxeter et al. (1986). The topic of tessellation itself, or of tiling, is popular in Logo projects even for beginners. It is discussed from several points of view by Abelson and diSessa (1981), Thornburg (1984), Kenney and Bezuszka (1987), and Clayson (1988). Brian Harvey's monumental project of raising the level of Logo users through and above intermediate level is published in Harvey (1985, 1986, 1987).

Constructionist learning, particularly Logo as a specific environment for such learning, is discussed in Solomon (1986) and Schuyten (1989), and in all the recent publications on this subject written by the staff of Media Laboratory at MIT.

Some preliminary data about the reactions of children to this microworld of Escher's stamp game can be found in Hadass, Movshovitz-Hadar, and Give'on (1990). The development of this microworld and the ongoing research on its educational merits have been carried out within Project Mass-Mathics, a project dedicated to mathematical education of low achievers. More information about this project may be found in Movshovitz-Hadar (1989).

A thorough analysis of the potential value of environments in which programs are used as manipulatives can be found in Perl (1990).

Y. S. Give'on
Beit Berl College, Kfar Saba, 44905, Israel
Nitsa Movshovitz-Hadar
Technion, Haifa, 32000, Israel
Rina Hadass, Oranim
University of Haifa, Tivon, 36910, Israel

References

Abelson, H., & diSessa A. (1981) *Turtle geometry: The computer as a medium for exploring mathematics.* Cambridge, MA: M.I.T. Press.

Clayson, J. (1988) *Visual modeling with Logo: A structural approach to seeing.* Cambridge, MA: M.I.T. Press.

Coxeter, H. S. M. et al. (1986) *M. C. Escher: Art and science.* Amsterdam: Elsevier Science Publishing Co.

Ernst, B. (1976). *The magic of M.C. Escher.* New York: Ballantine.

Hadass, R., Movshovitz-Hadar N., & Give'on Y. (1990) The cognitive challenge involved in Escher's potato stamps microworld. In *Proceedings: 14th PME Conference, Vol. II* (pp. 243-249). Mexico.

Harvey, B. (1985). *Computer science Logo style, Vol. 1 :Intermediate programming,* Cambridge, MA: M.I.T. Press.

Harvey, B. (1986). *Computer science Logo style, Vol. 2: Projects, styles, and techniques.* Cambridge, MA: M.I.T. Press.

Harvey, B. (1987). *Computer science Logo style, Vol. 3: Advanced topics.* Cambridge, MA: M.I.T. Press.

Kenney, M.J., & Bezuszka S.J. (1987). *Tessellations using Logo.* Palo Alto, CA: Dale Seymour Publications.

Movshovitz-Hadar, N. (1989). *Mass-mathics.* Jerusalem Convention on Education, Jerusalem.

Perl, T. (1990). Manipulatives and the computer: A powerful partnership for learners of all ages. *Classroom computer learning, 10,*(6).

Schuyten, G. (1989). Constructivism as a theoretical framework for Logo-based environments. In *Proceedings of the Second European Logo Conference (pp. vii-xxii).* Gent, Belgium.

Solomon, C. (1986). *Computer environments for children: A reflection on theories of learning and education.* Cambridge, MA: M.I.T. Press.

Thornburg, D. D. (1984). *Exploring Logo without a computer.* Menlo Park, CA: Addison-Wesley.

Zazkis R., & Leron U. (1990). Implementing powerful ideas—The case for RUN. *Logo Exchange, 8*(8), 11-14.

Higher-Level Math Thinking: Part II

by Douglas H. Clements

The subtle title indicates that there was indeed a Part I to this article—see RHOMBUS MADNESS in the Winter, 1991, issue of *Logo Exchange*. If you haven't yet studied that column,[1] you may wish to do so. We presented concrete examples and the researchers' theories. Table 1 provides a brief overview.

Table 1
Overview of the Theories

van Hiele's Levels of Geometric Thinking

Visual: The student sees figures as "wholes" and is unable to analyze their properties.

Descriptive: The student describes the properties and relations of figures and can use them in inductive arguments.

Skemp 's Ways of Understanding

Instrumental mathematics: "rules without reasons."

Relational mathematics: "knowing both what to do and why." Building up conceptual structures from which a student can produce an unlimited number of rules to fit an unlimited set of situations.

SOLO Taxonomy

Prestructural: Learners do not engage in the task or respond inappropriately, e.g., POLY 40, when POLY is defined as

```
TO POLY: TIMES :SIDE :ANGLE
REPEAT :TIMES [FORWARD :SIDE
  RIGHT :ANGLE ]
END
```

Unistructural: Learners are able to use only one piece of information, e.g., POLY 50 50 50

Multistructural: Learners are able to use several pieces of information but can't relate them, e.g., POLY 20 30 40 (different numbers but no meaningful relationship between the numbers and their role in POLY).

Relating: Learners integrate the separate pieces of information to produce a variable solution to the task, e.g., POLY 4 50 90

We also discussed the goals of the Atlanta–Emory Logo Project—breaking the vicious cycle of rote learning through Logo (Olive, 1991). We showed that some of the ninth-grade students had progressed to a fairly high level. What about the rest of the students? What levels did they achieve? How did the levels in one theory relate to those in another? What implications does all this have for teaching?

More Results

Students' Progress Through SOLO Levels

Most students started their Logo programming explorations Unistructurally and then progressed through Multistructural to Relating responses. The same pattern, however, did not hold for responses to the geometric concepts. Most students (18 of 30) only achieved Multistructural responses.

Now let's see how the levels related to each other. Olive measured students' SOLO level, van Hiele level, and quality of understanding (Skemp's instrumental vs. relational understanding). He measured them both in Logo programming and in geometry achievement. Were all these ways of thinking related?[2]

SOLO Level in Logo <—> SOLO Level in Geometry

The first question was whether students' SOLO level in Logo programming was related to their SOLO level in geometric tasks. Only students who obtained a Relating SOLO level response to the Logo programming tasks obtained a Relating SOLO level on the geometric tasks. Some students, however, obtained a Relating level in Logo but did not obtain it in geometry. Therefore, a Relating level response to Logo programming would appear to be necessary but not sufficient for obtaining a Relating level on the geometric tasks.

van Hiele Level in Logo <—> van Hiele Level in Geometry

Similarly, a van Hiele descriptive approach to Logo programming tasks appears to be necessary but not

[1] I would personally be shocked.
[2] There are many possible relationships coming up! I hope you enjoy seeing patterns emerge as you peruse them all. The startling truth, however, remains that *I will never actually know if you skip right to the discussion.*

sufficient for students to take a descriptive approach to geometric tasks.

van Hiele Level <—> SOLO Level

Things now get more complicated as we try to connect one theory to another. For example, students who take a descriptive approach to the Logo programming tasks tend to respond to these tasks at a Relating SOLO level. This was not symmetric, however. There were students at the Relating SOLO level who did not take a descriptive approach.

Similarly, students who take a descriptive approach to the *geometry* tasks responded to these tasks at the Relating SOLO level.

Thus, there seems to be a connection between the van Hiele approach and the SOLO level achieved. Students who approach a task *descriptively* are likely to achieve a Relating level. However, if they do not approach the task descriptively, they still might achieve a Relating level using a visual approach.

van Hiele Level <—> Skemp's Qualities of Understanding

First, if students did not achieve a Relational understanding of Logo, they did not understand geometric conceptions Relationally. They achieved only Instrumental understanding. This is parallel to what we just saw.

Second, the relationships are also similar. Students who take a van Hiele descriptive approach also achieved Relational understanding. In other words, students who approached Logo tasks by thinking about the *properties* of the figures also thought about the properties of figures in geometric tasks—and they showed a Relational understanding of both types of tasks.

SOLO Level (<—> van Hiele Level) <—> Skemp's Qualities of Understanding

Achieving the Relating SOLO level does *not* guarantee Relational understanding. Students had to approach the tasks descriptively to achieve Relational understanding. Therefore, using the descriptive van Hiele approach seems critical.

Skemp's Qualities of Understanding<—> Mathematics Grades

There was a weak, but positive, relationship between students' algebra and geometry grades. More interesting, *all* students who showed Relational understanding did better in geometry than their algebra scores would have predicted. The opposite was true for those with only Instrumental understanding. Overall, Relational students achieved a higher geometry grade than did the Instrumental students.

Whew! Time for a...

Discussion: What Does It All Mean?

The pattern of results is satisfying, if not surprising. Students had to program in Logo successfully and with some sophistication to achieve success and sophistication in geometric achievement. Such success with Logo did not *guarantee* success with geometry, but it was a prerequisite.

What *is* surprising is that three students did *not* achieve Relational understanding of geometric concepts. They apparently could not relate their previous geometric experiences to their work with Logo. This lead to *Recommendation 1*: Logo experiences need to be more directly linked to students' geometric experiences in the earlier grades.[3]

Another pattern involves the importance of students using a descriptive approach. They reach higher levels of learning when they think about the properties of geometric figures. Even when they achieve the SOLO Relating level, they are more likely to achieve Relational understanding if they use a descriptive approach. For students using a descriptive approach, the language they use—Logo and natural language (English)—*signals* the presence of important properties. Thus, they automatically have many ideas to use in solving problems, such as the measure of a figure's angles or the parallelism of its sides. They can then more easily build connections between the figures, their properties, and Logo programming (Relating SOLO level). This leads to a *Relational understanding* of the geometric concepts—knowing both what to do and why. The concepts will become part of an interconnected network of ideas.

We must ask a question: Why don't some students achieve higher levels? One reason is that using Logo can reinforce visual, instead of descriptive, approaches. The turtle provides immediate visual feedback. Using this feedback, students can become successful problem solvers at the visual level. I have raised this point continuously in this column; see especially "Strategies for Solving Turtle Geometry Problems" in the December/January 1990-1991 issue of *Logo Exchange [9*(4), 32-34]. This lead to *Recommendation* 2: Teachers must build challenges into Logo activities that encourage descriptive approaches. These activities should create what van Hiele describes as a "crisis in thinking" at the visual

[3] Olive is kind enough to state that our work (Clements & Battista, 1988; Clements & Battista, 1990) has shown that when such links *are* made, the Logo experiences can aid students' construction of key geometrical concepts (Battista & Clements, 1991). I was too humble to put this is the regular text. Of course, many people actually pay *more* attention to footnotes.

level, showing the deficiency of visual approaches and the power of descriptive approaches. Together with appropriate teacher interventions, such challenging tasks may be critical for optimizing learning with Logo.

Recommendation 3 deals with just these teaching responsibilities. Teachers could examine students' responses using the frameworks in Table 1. These frameworks could help them pose tasks and provide interventions that they tailor to the needs of a particular group of students.

Many students who learned algebra Instrumentally maintained a preference for instrumental learning in their Logo course. *Recommendation 4:* For such students, teachers should design Logo activities that generate students' desire for Relational learning. These activities should stimulate curiosity that was suppressed too frequently in the past. Students may need to be shown how to explore mathematical ideas.

Recommendation 5: Teachers and students should strive together for Relational understanding.

Recommendation 6: Logo work should be an important part of regular mathematics instruction, not just an add-on.

For more information on the project, contact

John Olive
Department of Mathematics Education
105 Aderhold Hall
Athens, GA 30602

References

Battista, M. T., & Clements, D. H. (1991). *Logo geometry.* Morristown, NJ: Silver Burdett & Ginn.

Clements, D. H., & Battista, M. T. (1988, November). *The development of geometric conceptualizations in Logo.* Paper presented at the meeting of the International Group for the Psychology in Mathematics Education—North American Chapter, DeKalb, IL.

Clements, D. H., & Battista, M. T. (1990). The effects of Logo on children's conceptualizations of angles and polygons. *Journal for Research in Mathematics Education, 21,* 356-371.

Olive, J. (1991). Logo programming and geometric understanding: An in-depth study. *Journal for Research in Mathematics Education, 22,* 90-111.

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

Douglas H. Clements, associate 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. He is currently working with several colleagues on a second NSF-funded project to develop a full K-6 mathematics curriculum featuring Logo.

Douglas H. Clements
State University of New York at Buffalo
Department of Learning and Instruction
593 Baldy Hall
Buffalo, NY 14260
CIS: 76136,2027 BITNET: INSDHC@UBVMS

Global Logo Comments

by Dennis Harper

Logo Exchange Continental Editors

Africa	**Asia**	**Australia**	**Europe**	**Latin America**
Fatimata Seye Sylla	Marie Tada	Anne McDougall	Harry Pinxteren	José Valente
UNESCO/BREDA	St. Mary's Int. Sch.	Monash Univ.	Logo Centrum Nederland	NIED
BP 3311 Dakar	6-19 Seta 1-Chome	6 Riverside Dr.	P.O. Box 1408	UNICAMP
Senegal, West Africa	Setagaya-Ku	East Kew 3120	BK Nijmegen 6501	13082 Campinas
	Tokyo 158, Japan	Victoria, Australia	Netherlands	São Paulo, Brazil

During last summer's NECC conference in Phoenix, I was a member of a panel whose purpose was to discuss whether Logo was entering the schools as a "Trojan Horse." This Trojan Horse concept was conjectured by Professor Seymour Papert some years ago. Looking at this month's submissions by our global field editors, one could certainly make a supporting case for the Trojan Horse theory.

In Australia, Anne McDougall finds Logo going beyond the introductory stage and sees Logo thinking entering many areas of their educational system. In Japan, Marie Tada explains how Logo is acting as a front end to school multimedia projects. Fatimata Seye Sylla reports that in Senegal, West Africa, Logo becomes just another activity in a summer camp for youngsters.

Logo at ACEC '91

by Anne McDougall

The Ninth Australian Computers in Education Conference was held at Bond University of the Gold Coast in Queensland from September 22 to 25, 1991.

Scanning the conference program might at first have been alarming for Logophiles because only 3 of the 64 sessions had any reference to Logo in the titles! Does this mean the development of Logo thinking and use by Australian teachers and students has almost come to an abrupt halt? Fortunately, it does not, but it does indicate, I think, an interesting development in Logo adoption and practice in this country. More on this in due course, but first to the three papers that clearly were about Logo work.

Two of these papers were from Coombabah Primary School in Queensland, where a group of 60 children in Grade 6 are participating in a project using *LogoWriter* and a high density of computers (half the children have laptop machines). This project is sponsored by the Queensland Education Department.

Jenny Betts, one of the teachers involved in the project, presented a session titled "The Creation of Databases Using *LogoWriter*." She described some interesting database development work done by the children, using a variety of approaches, and becoming, as they do this work, highly skilled programmers. Jenny noted that the children's main source of programming ideas is each other, although other assistance comes from experts visiting the school, Logo books and the *LogoWriter* manual. She described her role as a teacher as that of a guide "mostly to another child who can answer the question." She went on to emphasize the importance of the availability of large amounts of time for students to work on these projects.

Another Coombabah teacher, Dave Mitchell, presented some *LogoWriter* mathematics activities in which children used the page for noting estimated answers before performing calculations in the Command Center.

The third presentation with Logo in its title was from Gary Stager, currently back in Australia for some teacher professional development workshops and other engagements. In "New Environments for Intellectual Expression: Mac *LogoWriter* and *Logo Ensemble*," Gary demonstrated many of the distinctive features of these new products. His talk interested those in the audience already familiar with Logo and *LogoWriter* because it involved the increased potential for expression through programming in these new environments. On the other hand, Gary's talk rather stunned some participants who were not already familiar with *LogoWriter*, but he entertained us all with his virtuoso programming and impressive presentation skills.

So, what then of Logo and the rest of the conference? Previous conferences of this kind usually have included quite a group of papers describing classroom implementation or research with Logo; in some years there even has been a delineated Logo "stream" within the conference.

Despite the impression created by the titles of the papers, there was a good deal of presentation and discussion of Logo ideas at ACEC '91, although the ideas were more integrated into papers focusing less directly on implementing and "evaluating" Logo itself. It is as if, at last, the language is beyond the introduction phase; now Logo "thinking" is contributing to the discussion of many other educational computing issues and practices. The following are just some examples.

- Chris Bigum, in his paper "Schools for Cyborgs: Educating Aliens," made extensive reference to the writings of Papert and Turkle.

- Some of the metaphors deliberately used in Logo contexts were considered in Carolyn Dowling's paper "Metaphorically Speaking—The Language of Classroom Computing."

- In his paper "Computers, Children, and Learning: Some Predictions for the 1990s," Peter Shackleton predicted that Lego/Logo and computer control technologies would become increasingly important for problem solving and concept development.

- Papert and Turkle were extensively quoted in a fascinating paper and most entertaining presentation entitled "Editable Selves: A Thought Experiment in Information Technology for Thinking Teachers" by Neville Stern.

- Logo research work provided part of the context for a methodological paper, "Learning With Computers: What Type of Research?" by Helga Rowe.

- We learned more about the *LogoWriter* work at Coombabah Primary School from Greg Grimmett's paper "Laptop Computers: Panacea or Problem?"

- There was also a paper entitled "Searching for Solutions: Experimental Applied Mathematics in a Primary School Environment," by John Belward, Dave Mitchell, and Michael Ryan.

- A paper on language study, "Unraveling Rules and Exceptions," by Karen Hallett, rounded out the presentations.

The establishment of OzLogo, a Special Interest Group for teachers interested in the use of Logo in primary, secondary, and tertiary education, was announced at ACED '91. This newly formed group, based in Victoria but catering members throughout Australia, will arrange meetings and teacher professional development activities, provide a mutual support network for Logo-using educators, and produce a regular newsletter or journal.

Copies of the *Proceedings of the 1991 Australian Computers in Education Conference* may be obtained from the Computer Education Group of Queensland, GPO Box 1669, Brisbane, Queensland 4001, Australia, for Aus$25 plus postage.

From Asia

by Marie Tada

Konnichi wa (Hello!) from Tokyo.

So much has been going on with conferences, interviews, and professional contacts that it's hard to know where to begin. I felt fortunate to receive an invitation from the Czech Ministry of Education to participate in the Third East/West Invitational Seminar on New Technologies in Education (jointly sponsored by the Czech Ministry of Education and the International Center for Technology and Education at the Institute of New Technologies, Moscow, USSR and University of Hartford, USA). This turned out to be a unique gathering of international educators and researchers and was an even more intense experience because it coincided with the coup in the USSR. Many of our delegates were from these regions, and in the ensuing discussions and encounters we learned much as a group about the difficulties of living under repressive regimes. I left the conference with a deep respect for the people who are sometimes invisible behind the facades of governments that do not run by the democratic rules we take so much for granted.

The conference left me with a lot to think about. Many constructive cooperative projects were discussed—especially in the area of telecommunications. One thing missing, aside from parts of my presentation and some ideas presented in discussions, was Logo. To me, Logo is a natural base for creating a technology-using culture in a school while allowing children to explore and discover in the Logo learning environment. It seems, however, that Logo is perceived as being too teacher-education and equipment intensive, and there are perhaps too few cases where children get beyond the fundamentals of turtle graphics. I feel that Logo-based computer studies can pave the way and create a mind set for the entrance of more Logo-like uses of technology in the schools—telecommunications units and projects using hypermedia and multimedia being a few of these.

The Prague conference was a rich and meaningful experience for all involved. Despite the great changes going on now in those countries, we hopefully will remain aware of the remaining economic and societal difficulties our Soviet and eastern and central European colleagues may face in bringing technology-based international exchanges into the schools. From such awareness, perhaps we can build bridges of helpful assistance.

In the past year, I have become very much involved with multimedia. I feel that perhaps there has been too much emphasis on the use of many kinds of hardware (like laser disc players and CD-ROM) and not enough emphasis on what multimedia means—the manipulation of graphics, text, animation, video, and audio by the computer. If we think in terms of this latter definition, then Logo certainly has been giving us many elements of multimedia all along with lots of powerful ideas thrown in to boot!

As a case in point, I recently had the opportunity to talk with people at the New Media Laboratory at Fukutake Publishing Company. Last year, I wrote a *Logo Exchange* article about *Findout*, Fukutake's excellent Logo-based software (which is primarily geared to the elementary and junior high levels but could be very effectively used at higher levels as well). *Findout* offers word processing, database, graphics, music, animation, and some telecommunications functions.

I became fascinated with the number of interfaces that could be used with *Findout* and thought how we are certainly on the road to making a fully multimedia Logo. I observed the *Findout* program controlling the NEC COMBOY video tape player, allowing video sequences to be searched for and shown as part of the Logo project. Because a laser disc player can be attached to the RS 232C port, it should also be possible to control the laser disc with *Findout*.

I took pictures using a Fujix still video camera, which allows up to 50 shots to go directly onto a floppy disk and then be shown directly on a television screen with the use of a special adapter. We then used *Findout* to sequence and build a story around these shots. An FM sound board used with the *Findout* Harmony program allows construction of up to six-note chords in eight octaves. The music that can be easily created is quite impressive.

An AD-232 adapter allows measurements to be made and manipulated in *Findout* for heat, water levels, movement, voltage, and so forth. Much of what the project's science and math teachers have done with these sensors are collected on "library disks" and sent out as resources to other teachers. The Valiant turtle can be emulated, and a board can be used that will input the text of words, match these up with a dictionary, and give a voice output of the Japanese (and some English) word equivalents. It seems to me that it won't be long before we are inputting all kinds of digital data to our Logo programs as Logo continues to evolve as a timely modern tool for learning and exploration.

At Fukutake, I also had the chance to view some of the entries in their annual contest for good ideas in using *Findout*. There are three contest areas—school applications, programming, and sound and graphics. One programming award went to an English teacher who made a program that simulated a hamburger stand. Children could input information on how much they wanted to spend and what they wanted to buy. Many basic phrases, such as "What would you like?" and "That will be 300 yen," brought together interesting English learning with a good Logo programming idea.

Another winner was a junior high school boy whose program allowed the user to choose an adventure to go on with a dog—to the beach, the art gallery, and so forth. Another was a take-off on "Where's Wally," with the user inputting the number of turtles to be displayed and then inputting commands. Only one turtle will follow the commands correctly, and it is up to the user to discover this "Wally" from among the errant turtles. A teacher created an art project that constructed a variety of solid figures in wire-like 3-D form and asked the user to apply gradations of color to fill in and complete the picture. As a good incentive to participate in the contest, the winners get some great prizes (personal computers, printers, and scanners) as well as the satisfaction of winning. The results are then compiled and sent to other *Findout* users—a fine resource for sure.

Africa Revisited

by Fatimata Seye Sylla

I would like to remind our readers that the "Laboratoire Informatique et Education (L.I.E.)" was the Logo research project implemented in Dakar, Senegal, in March, 1982. This study aimed at investigating the learning process in a Logo environment, the impact of the computer and Logo on teachers and school children, and interactions in the classroom.

During this past annual school break, July through September, the Computers and Education Project of Senegal undertook several activities to promote the use of computers by school children and adults nationwide. And, of course, Logo was the software used.

For several weeks before the break, the project team had to organize training courses for the future trainers selected within the so-called "popular educators" of the Ministry of Youth. Within the 10 regions of Senegal, the Ministry of Youth, in collaboration with the Ministry of Technology, set up camps with computers and software (mainly Logo and computer games). Government and private institutions contributed financially to send their personnel's children to the camps.

Children had a good time drawing, playing music, and doing mathematics using Logo on the floor with their feet and on the screen with the turtle. In addition to doing Logo activities, they played other Senegalese and Western games. They swam, sang, danced—they had fun!

What was impressive in these camps was that the computer played a major role in allowing children to get acquainted with each other. One child would admire another's drawing, ask him how he did it, and that would be a start of a new friendship.

In Dakar, the Youth Foundation sponsored and actively participated in another special event. Adults and young children were trained in the use of computers according to each person's specific need. During the morning sessions adults learned how to use word-processing programs, and children programmed in Logo in the afternoons. This went on for three weeks from September 10 to September 30, 1991, at the Blaise Senghor Centre. Twelve secretaries were trained right away.

These kinds of activities aim at spreading the use of computers not only within the schools but also outside the school system. Thus, more training courses are organized within the Laboratory for trainers who would be ready to train children and or adults throughout the country.

At the same time, the teacher training curricula now includes computer training courses as the use of computers in the Senegalese educational system continues to grow. Computers have been shown to have a positive impact on the interaction between teachers and students and on the way knowledge is transmitted and acquired.

This short report shows how the objectives of the former Logo project have evolved from studying the impact of the use of Logo and computers on the school system to the widespread use of computers by the Senegalese people.

Dennis Harper
University of the Virgin Islands
St. Thomas, USVI 00802

LONG DISTANCE LOGO

Educators—Let the classes come to you!
Available for *LogoWriter* and Terrapin's *Logo PLUS*

Introduction to Logo For Educators, a graduate level ISTE *Independent Study* course. Learn at your own pace while corresponding with your instructor by mail, fax, or telecommunications. Tuition is $349.00 for a four quarter-hour class. Tuition is reduced for non-credit enrollment.

WORK INDIVIDUALLY OR WITH A GROUP

Take ***Introduction to Logo For Educators*** at home, or study with a group of colleagues. The course uses video tapes (ON LOGO) with MIT's Seymour Papert, printed materials, textbooks, and disks. View the tapes, read and report on course materials, do projects, and design lessons for students.

NOT JUST ANOTHER CLASS

Dr. Sharon Yoder, editor of *Logo Exchange*, has designed ***Introduction to Logo For Educators*** to provide staff development and leadership training. The four quarter-hour course carries graduate credit from the Oregon State System of Higher Education.

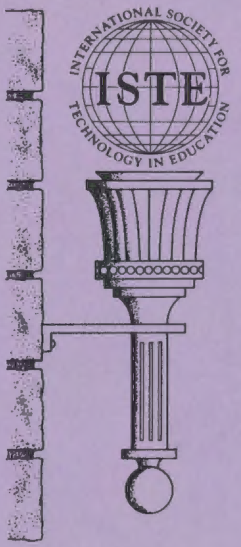
ON LOGO VIDEO TAPES

School Districts may obtain a license for the use of the ON LOGO package of 8 half-hour videotapes and 240 pages of supporting print. For a one-time fee of $1,295, the package may be obtained with both tape and print duplicating rights, enabling districts to build libraries at multiple sites. Tuition rates are also lower for group enrollment.

To receive an Independent Study course catalog, write or call:

International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Phone: 503/346-2412 Fax: 503/346-5890
Bitnet: ISTE@Oregon CompuServe: 70014,2117
ISTE.Net (GTE): ISTE.office

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.

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.

ISTE
1787 Agate St., Eugene, OR 97403-1923.
ph. 503/346-4414.

Logo Exchange
ISTE
1787 Agate Street
Eugene, OR 97403-1923