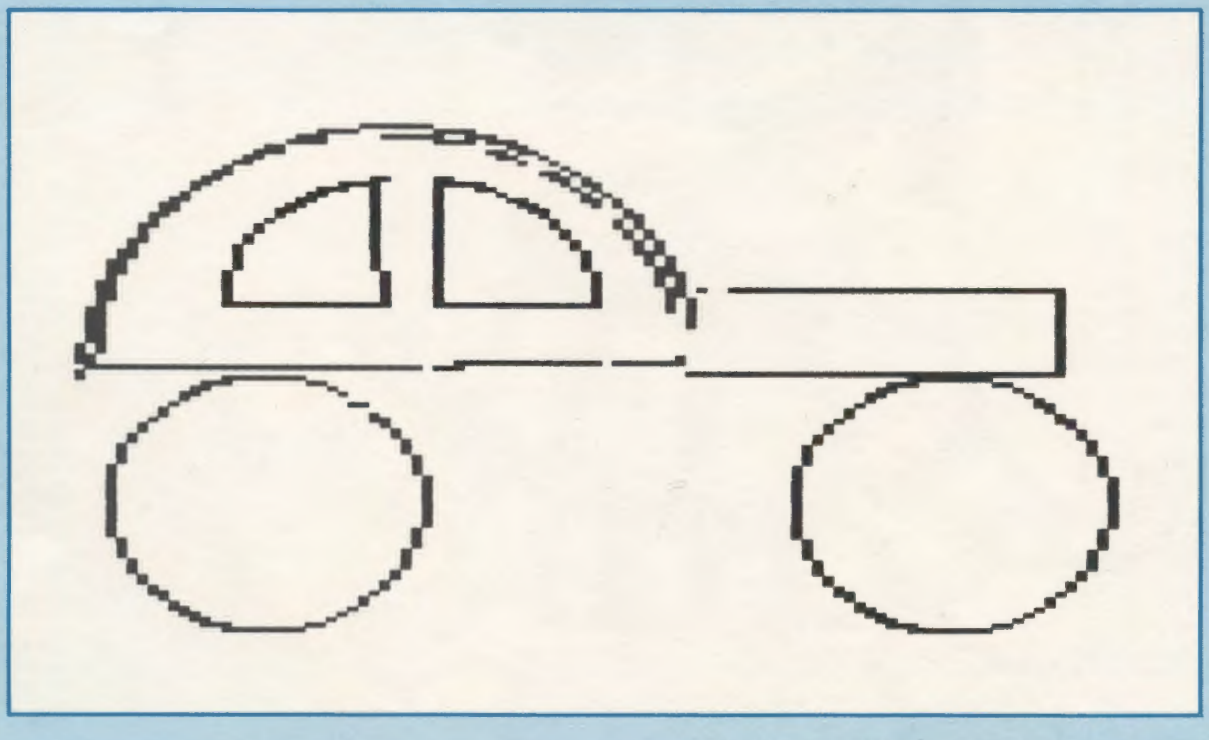
Journal of the ICCE Special Interest Group for Logo-Using Educators

LOGO EXCHANGE

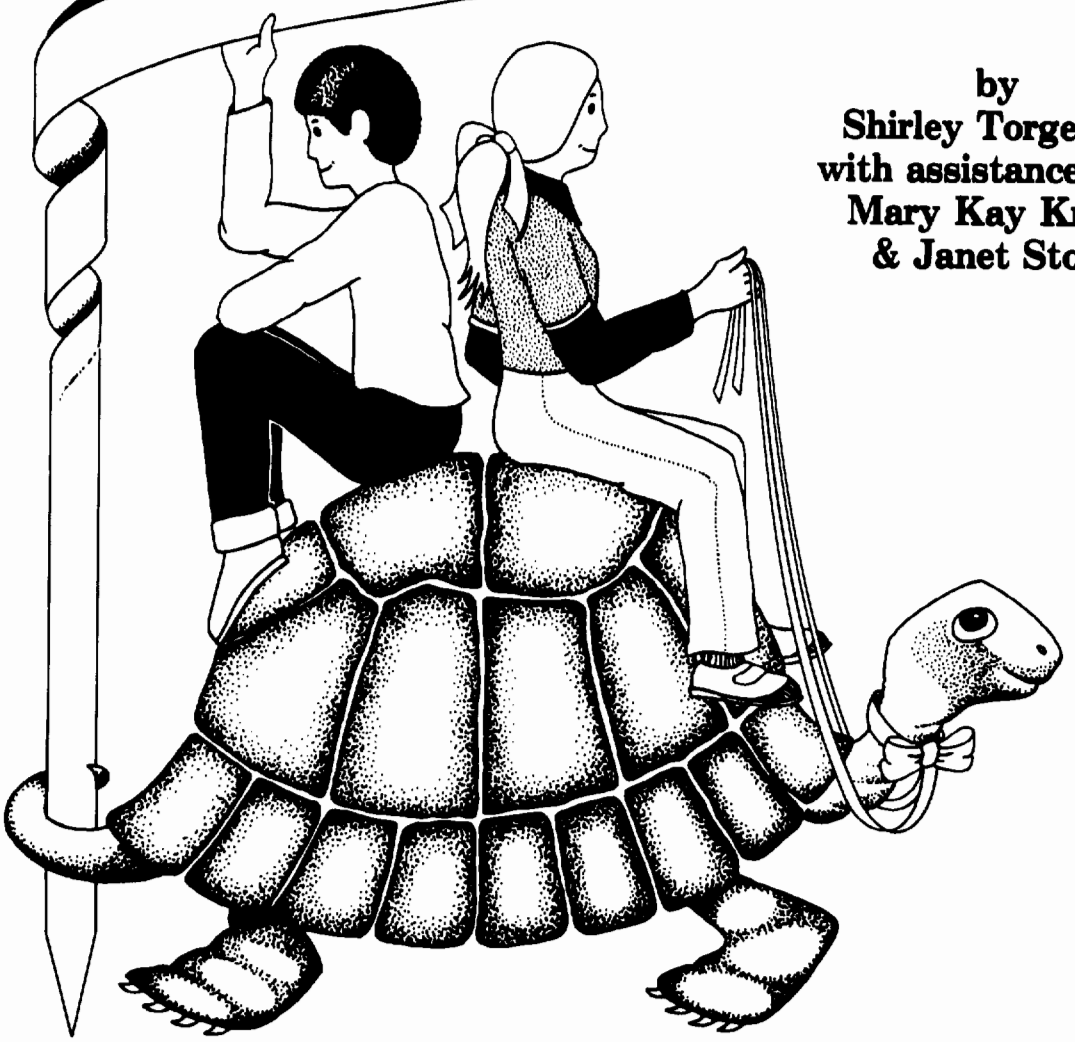
MARCH 1988 VOLUME 6 NUMBER 7



International Council for Computers in Education

ICCE Publications

Logo in the Classroom



by
Shirley Torgerson
with assistance from
Mary Kay Kriley
& Janet Stone

Logo in the Classroom integrates Logo into your elementary curriculum. Twenty lessons were developed in a classroom setting as a response to "How can Logo work in a classroom where computers are in short supply?"

Detailed teacher information is given for each lesson along with copyable practice sheets, transparency masters and 12 charts. Useful for teacher inservice.

$13.00 (US) plus $2.50 shipping.

INTERNATIONAL COUNCIL FOR COMPUTERS IN EDUCATION
ICCE
University of Oregon 1787 Agate St. Eugene, OR 97403

LOGO EXCHANGE

VOLUME 6 NUMBER 7 Journal of the ICCE Special Interest Group for Logo-Using Educators MARCH 1988

Logo Exchange is the journal of the International Council for Computers in Education Special Interest Goup for Logo-using Educators (SIGLogo), published monthly September through May by ICCE, University of Oregon, 1787 Agate Street, Eugene, OR 97403-9905, USA.
POSTMASTER: Send address changes to Logo Exchange, UofO, 1787 Agate St., Eugene, OR 97403.

CONTENTS

SIGLogo Membership (includes *The Logo Exchange*)

	U.S.	NON-U.S.
ICCE Member	24.95	29.95
Non-ICCE Member	29.95	34.95
ICCE Membership (includes *The Computing Teacher*)	28.50	31.50

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

The *Logo Exchange* is produced on a Macintosh SE and Laser Writer donated by Apple Computer, Inc. (LOCAL "PAGE)

From the Editor

Logo is Dead...Long Live HyperCard?

Wait! Don't stop! Read on...

Each fall and spring, computer conferences seem to cluster into a few short months, keeping some of us "on the road" for weeks at a time. I have just returned from attending a series of such conferences. At each of these conferences, someone said to me "I hear that Logo is pretty much dead." While they don't continue with "but HyperCard lives," the flavor is still there: HyperCard sessions are packed, those who haven't tried HyperCard are apologetic, and one can overhear much chatter between sessions and in exhibit areas about the wonders of HyperCard. Even the *Logo Exchange* has included several articles about HyperCard.

The "hype" about HyperCard reminds me of some of the things that we have heard over the years about Logo. To listen to the chatter, one would think that HyperCard is going to solve the problems of education, not to mention the problems of the world. Although HyperCard is a fascinating product with a great deal of potential, I find myself wondering if the "HyperCard community" is going to turn this new software into a "cause" in much the same way as the Logo community has at times "oversold" the virtues of Logo.

Research on Logo is simply not confirming many of the glowing claims made by Logo leaders and Logo teachers. Some feel that most of these studies are flawed or that they fail to examine facts that we as Logo teachers know are important. Nonetheless, they *are* affecting the educational community's attitude towards Logo.

Why hasn't Logo fulfilled its promises? Why aren't more people using Logo? Why aren't research studies confirming the value of Logo? Certainly, no one has easy answers to these complex questions. However, in this issue of *LX* we are taking one small step towards addressing one of these concerns. A frequently stated reason for teaching Logo is that it will enhance the problem-solving abilities of its users. Few research studies bear this out. The article "Powerful Ideas in Problem Solving and Logo" addresses this issue and attempts to point towards a solution. We hope you will find it of interest.

Is Logo dead while HyperCard lives? I doubt it, and most likely so do you loyal readers of *LX*. But what *should* Logo become? More like HyperCard? More powerful? Faster? Broader (e.g., Lego Logo)? More like existing applications packages? I'd be interested in your thoughts...let's get a "Letters to the Editor" column going.

Long live Logo!

Sharon Burrowes Yoder, Editor
The Logo Exchange
ICCE, University of Oregon
1787 Agate St.
Eugene, OR 97403-9905

Monthly Musings

Hyper Hype II

by Tom Lough

Last month, I wrote about my initial reaction to the HyperCard program recently released for the Macintosh. Since then, I have had the opportunity to explore this software in a little more detail. The more I worked with it, the more I sensed a kinship with Logo.

At first glance, HyperCard appears to be a sophisticated information manager which uses a stack of index cards as its organizational metaphor. Backgrounds, fields, and buttons can be combined with graphics to organize and present many different kinds of information. Users can control the environment of a stack by writing scripts with the HyperTalk language.

HyperTalk consists of English-like commands with reasonably sensible syntactical organizations. Thus, reading a HyperTalk script is not a totally befuddling experience. This self-documenting characteristic reminds me of Logo. [It is possible to put comments with a HyperCard script, nevertheless.]

The scripts perform in a manner similar to the WHEN demons of Atari Logo. In that particular implementation, Logo checks continuously the state of a specified condition, such as whether two particular turtles have collided or whether a particular turtle has encountered a line of a specified color. WHEN the condition is true, then a list of specified Logo instructions is carried out. Thus, a typical WHEN statement might be:

```
WHEN 17 [ PRINT "POW" TOOT 1 440 7 6]
```

Here, condition 17 corresponds to checking whether turtle #1 and turtle #3 have collided. WHEN this happens, Atari Logo prints "POW" and uses voice #1 to TOOT the tuning note A (440 cycles per second) at a medium volume (7 on a scale from 0 to 15) for 6/60ths (one-tenth) of a second.

Instead of using "WHEN," each HyperCard script begins with the word *on* and a specified condition, such as whether the mouse button has been pressed or released. This is followed by a set of HyperTalk instructions. When (or, *on* the occasion that) the condition is true, then the instructions in the script are carried out. A common script is the following:

Cover: Andy Houck, grade 3, is in the Resource Enrichment program at the Conover Road School in Colts Neck, New Jersey. He designed the cover art using the arc and circles on his own without using the tool procedures.

Monthly Musings — CONTINED

```
on mouseUp
   go to the next card
end  mouseUp
```

This script is activated by the mouse button. The mouseUp condition refers to a message sent by the computer to indicate that the button on the mouse, having been depressed, has been released. On that occasion, HyperCard then goes to the next card in the stack and displays it on the Macintosh screen.

Although the WHEN demons of Atari Logo might help one understand better the general operation of HyperCard scripts, there is a fundamental difference. The WHEN demons are global. That is, the WHEN command is given only once. Regardless of what Atari Logo is doing, it is always checking the specified condition, ready to carry out the associated instructions. In HyperCard, however, the scripts are local. That is, each script is associated with a particular object, such as a card, a stack, a background, a field, or a button. When an object is active, the scripts associated with it can carry out their instructions when the appropriate conditions are met. This allows different scripts to be written for different buttons, fields, or other objects.

In addition, there is an established hierarchy of objects which channels signals (or messages) upward for activation of scripts. This hierarchy has HyperCard itself at the top, and proceeds downward through the Home Stack, other stacks, backgrounds, and then cards. Both buttons and fields are subordinate to cards. Thus, if a button is clicked, then HyperCard searches first through the button's scripts to see if there is anything to do on mouseUp. If it finds an on mouseUp script, then it will carry out the instructions. For example, if the script above were found, then the next card would appear on the screen. If there is no on mouseUp script associated with the button, then HyperCard searches the scripts of the card, then the background, then the stack, the Home Stack, and finally HyperCard itself. At any point, if it finds an on mouseUp script, it would carry out the instructions and return to the user. If none is found, then nothing is done.

Although the local nature of the scripts and the consideration of the hierarchy were something I had to learn more about, the similarity between the script idea and the WHEN command of Atari Logo was somewhat comfortable for me. But I still felt incomplete. I wondered, "Is there anything in HyperTalk which is comparable to a Logo procedure?" As it turned out, there is: the HyperTalk function.

Although HyperTalk comes with many predefined functions [similar to primitive procedures], it is possible for you to define your own. Just type "function" followed by the selected function name and the names of any inputs. On the next lines, type the instructions, and finish up with "end" and the function name once again. Once defined, the function names can be used like other HyperTalk instructions. Here is an example adapted from Danny Goodman's *The Complete HyperCard Handbook* (Bantam Books) of a function to calculate and return the value of a factorial of an integer given as input:

```
function factorial n
   if n < 2 then return 1
   else return n * factorial (n - 1)
end factorial
```

The corresponding Logo procedure might look like this:

```
TO FACTORIAL :NUMBER
 IFELSE :NUMBER < 2
   [OUTPUT 1]
   [OUTPUT :NUMBER * FACTORIAL(:NUMBER - 1)]
END
```

Once again, the similarity with Logo helped me to understand the functions of HyperTalk. The general organization of the definition of each is about the same. The factorial example also shows the recursive ability of HyperCard (although when I used numbers greater than 17 or so for the input, I got an interesting error message, protesting "too much recursion.").

To call the Logo procedures, you could type:

```
PRINT FACTORIAL  5
120
```

Because the FACTORIAL procedure is an operation which provides an output as a result, you must provide a place for the result. Here, it was used as input to the PRINT command.

To call the HyperTalk function, you could type:

```
put factorial (5) in field 1
```

HyperTalk places the value of 120 into field number 1 of the specified card.

Logo procedures, like the WHEN statements, are stored in a global workspace, and are accessible from practically anywhere within Logo. On the other hand, HyperTalk functions, like the "on" scripts, are stored with particular objects (buttons, fields, etc.), and are local to those objects and those lower in the object hierarchy. Through this hierarchy, you can make functions available to specified groups of objects. For example, if you define a function which will be used from all levels of Hyper-

Monthly Musings — CONTINUED

Card, then you could associate the function with the Home Card.

Although the function capability of HyperCard is made for operations, it is possible to write functions which perform as commands.

The similarities between Logo and HyperCard have helped me to learn more about this exciting new computer tool, and I commend it to you. I believe that, with the attitudes and experiences developed during the past several years with Logo, many educators will be able to recognize and capitalize on the educational power of HyperCard. As always, I would be interested in your work.

```
FD 100!
```

Tom Lough
PO Box 5341
Charlottesville, VA 22905

LOGO CONNECTIONS:

A Two-Week Logo Retreat will be offered at the University of Virginia in Charlottesville, VA, August 1 - 12. The intent of the retreat is to bring together teachers across the country who are interested in connecting with each other and learning how to create "Logo Connections." From its beginning, Logo was designed as a language which could be connected to objects in the outside world, such as switches, motors, lights, and other sensors. In addition, Logo work has always emphasized the importance of the social side of learning through connections with colleagues and friends. Both of these aspects of connections will be emphasized at the retreat in an informal, hands-on environment.

The workshop faculty of Glen Bull, Gina Bull, Judi Harris, Tom Lough, and Cheryl Wissick have a wide range of interests and expertise, ranging from pendula in physics to switch inputs for handicapped users. Among them, they have written numerous books and several hundred articles on Logo.

The two-week retreat will be limited to a maximum of twenty participants. Air conditioned dormitory rooms will be available at summer rates for those who require housing. Three hours of graduate credit will also be available through the university, on an optional basis. A wide variety of fun-filled social experiences throughout the two-week period will emphasized the importance of personal relationships in teaching with Logo. Participants must have intermediate Logo programming skills prior to the retreat. For an application or more information, write to:

Logo Connections c/o; Tom Lough ;Curry School of Education; University of Virginia; 405 Emmet Street; Charlottesville, VA 22903 or call Peggy Marshall at 804-924-7471

Logo Ideas

Using SETH: When, Why and How?

by Eadie Adamson

SETH is a command which, together with a number, sets the turtle to an absolute direction. You can think of it as using compass points.

SETH 0 is *always* pointing to the top of the screen
SETH 90 *always* points directly to your right
SETH 270 *always* points directly to your left
SETH 180 *always* stands the turtle on its head

How and when should you teach it?

* * * * *

An important aspect of a good Logo environment is placing the student in control of his own learning. One of the reasons I really like *LogoWriter* is that, by putting a program on disk, the child can take over. Taking over, however, does not always bode well for success. There still may be countless problems to solve which are sometimes beyond a particular student's ability. It becomes important to provide simple tools or ideas which they can use as they encounter difficulties.

One tool which I find particularly useful to give students early in their learning is SETH, or SETHEADING (different versions of Logo allow you to use both terms, *LogoWriter* accepts only the abbreviation). SETH, or SETHEADING, changes the direction the turtle faces based on degrees. The orientation is directly related to compass points students may have already encountered in map-reading in their social studies classes: 0 corresponds to north, 90 to east, 180 to south, and 270 to west (see Figure 1).

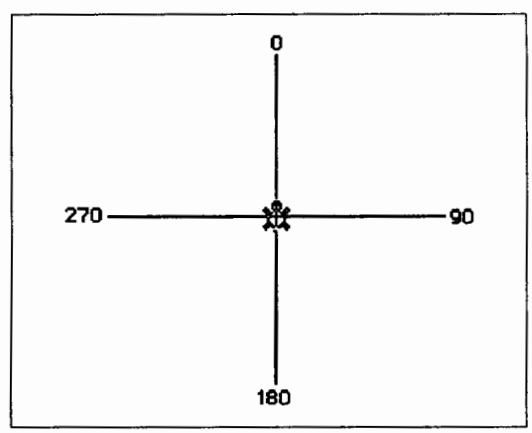


Figure 1

Logo Ideas — CONTINUED

Just as these directions are orienting directions on map, they are also important anchors for turtle graphics. SETH is a convenient tool to use when the turtle is "lost," since it is an absolute direction as opposed to the relative directions of LEFT and RIGHT. For example, suppose you've been turning the turtle, perhaps have accidentally typed RIGHT 990. If you now want the turtle to face straight up, type SETH 0.

Can teaching SETH early get in the way of learning the right-left commands? My own experiences with teaching SETH have indicated that if students have these anchor points to use, they will use them when necessary and that giving them just the four compass points does not really interfere with their learning about and using RIGHT and LEFT. It may, in fact, give their confidence a boost, since they know how to get themselves untangled when necessary.

When are children ready to learn SETH? As with most new Logo ideas, I believe they are ready *when they need it.* This can mean that even a kindergarten child might learn to use SETH. Several years ago, when I was working with kindergarten children in a fairly large and active class, Damaris was in the process of drawing a house. She needed to draw a roof. I couldn't stay with her while she worked everything out, so I made her a small drawing with SETH commands and arrows showing their direction. By the end of the class Damaris finished her house! That was not the end of the story, however. During the next class a week later, I suddenly noticed Damaris teaching her neighbors how to use SETH! Damaris not only used the tools I gave her, she remembered them because they were useful to her and proceeded to share the new information with her peers. (Perhaps this means that we as teachers need to look carefully at our decisions about when and what to teach. Are we holding back on teaching something a child can really use because of our own misconceptions about what that child can understand?)

It can be helpful to put little stickers around the monitor: 0 at the top, 180 at the bottom, 270 on the left, and 90 on the right. Once SETH has been introduced, these numbers act as a good reminder when children are *turtling* about the screen. If a child has typed RIGHT 990, for example, when the intended command was RIGHT 90, a quick switch to SETH 90 will solve the problem without need for teacher intervention. For those using SETH more actively, the numbers around the monitor serve as additional visible cues, from which the intervening directions may be derived. Setting the heading of the turtle somewhere between 0 and 90 will get you started for drawing a tree, perhaps.

When using shapes rather than the turtle (with *LogoWriter*, Sprite and Atari Logo, for example), SETH is extremely important for setting direction. SETH 90 will always turn the turtle to the right, even though the shape itself will not turn. It is also far less confusing to use SETH 90 than to start a procedure with RIGHT 90, only to run the procedure again and discover that suddenly your turtle, in whatever shape, is now heading to the bottom of the screen. SETH is more reliable!

If you use the LogoWriter Activity cards in your classes, I suggest you change the ones which move shapes (there are several) so that students are directed to use SETH 90 instead of RIGHT 90. This will save you—and your students—considerable distress.

Sometimes I give my students little diagrams which they keep in their notebooks. The diagram (see Figure 2) gives a number of useful directions which can be used with SETH (or with RIGHT if you place the diagram so that 0 is aligned with the current direction of the turtle). This is similar to many "Turtle-Turner" devices which are on the market, but the same little diagram can be used to produce transparencies which a child can place on the screen to help determine direction.

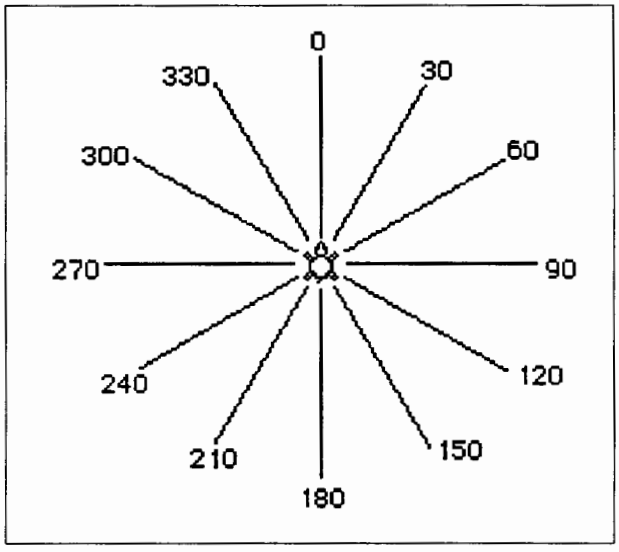


Figure 2

The diagram is a nice idea, but a better idea might be to help your students make their own. If you use *LogoWriter*, help them to construct one which has at least the four basic points of 0, 90, 180 and 270. Let them work out the others. You might start off with:

```
SETH 0
FORWARD 50
BACK 100
FORWARD 50
```

Finish the diagram, then LABEL the points. Challenge your students to make a diagram of their own which includes some intervening points as well. Let them print their own diagrams! Not only do they have a tool they can use, but they have

Logo Ideas — CONTINUED

ownership of the tool because they created it themselves. You can make transparencies for them from their own diagrams, too!

Another way I have introduced SETH is to ask my students to pretend the turtle can only respond to FORWARD, BACK and SETH—no RIGHT or LEFT allowed. Then I give them a few simple challenge designs to try using only these three commands (see Figure 3).

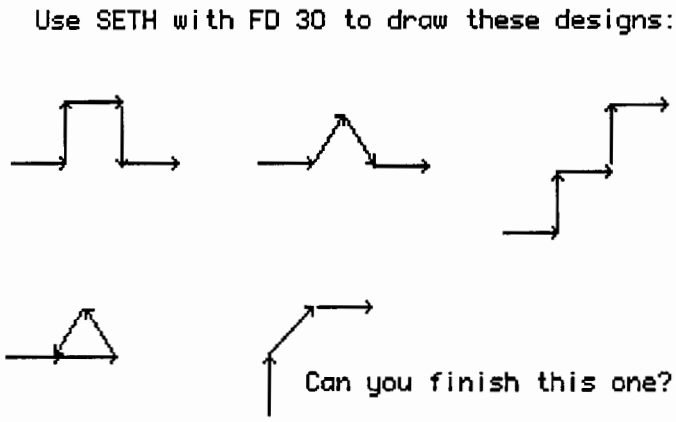


Figure 3

Some students may think of programming the four headings by their compass names. This is presents a good opportunity to teach about OUTPUT as well. Rather than programming EAST to be SETH 90, introduce OUTPUT so that EAST will OUTPUT 90.

```
TO EAST
 OUTPUT 90
END
```

Then SETH EAST becomes the command to use (and EAST can also be an input to LEFT or RIGHT if the student wants to use it that way!). Not only does this make a more direct connection with compass points learned elsewhere, but it will help the student remain aware of what is actually happening when EAST is used, namely setting the heading of the turtle. Since SETH is a primitive, I think it's important not to allow children to lose sight of it by burying it in a procedure. Encourage them to use it instead!

How to explain OUTPUT? One simple approach which I have used with my fourth grade classes is to hand someone a disk jacket. Since OUTPUT passes something on to be used in another command or procedure, ask the student to OUTPUT the jacket to his neighbor. Make it a simple demonstration in which each OUTPUTS to the next until it gets back to you, when you—finally!—*use it* to store a disk. The analogy may not be perfect, but it gets across the idea of passing something on, which is precisely what OUTPUT does. Then suggest writing the procedures to OUTPUT the heading. Perhaps someone will think of OUTPUT as useful for shapes and colors too. For example:

```
TO BLUE
 OUTPUT 5
END

TO CAR
 OUTPUT 26
END
```

allows you to type SETC BLUE and SETSH CAR. SETH EAST, after EAST has been written, will set the car in the correct direction for moving across the screen.

If your students are already using inputs, pose this problem:

1. Write a square procedure with inputs. Use only SETH to turn.
2. What can you do with inputs so that you can use SQUARE to do a SPINSQUARES?

One might think this is an insoluble problem, but it can be done by adding an input, INCREMENT, to SETH: SETH 90 + :INCREMENT. Don't forget to add it to the name of the SQUARE and SPINSQUARE procedures as well.

If you want to try this, it might look something like this:

```
TO SQUARE :SIZE :INCREMENT
 SETH 0  + :INCREMENT
 FD :SIZE SETH 90 + :INCREMENT
 FD :SIZE SETH 180 + :INCREMENT
 FD :SIZE SETH 270 + :INCREMENT
 FD :SIZE
END

TO SPINSQUARES :SIZE :INCREMENT :ANGLE
 SQUARE :SIZE :INCREMENT
 SPINSQUARES :SIZE (:INCREMENT + :ANGLE)
      :ANGLE
END
```

SQUARE 50 0 will simply draw a square; SQUARE 50 45 will draw a square turned on its tip; SPINSQUARES 50 45 45 will spin the squares! (The increment and the size inputs are passed to SQUARE; ANGLE is the turn for SPINSQUARES.) Now, how about adding a STOP rule for SPINSQUARES?

Eadie Adamson
Allen-Stevenson School
132 E. 78th St.
New York, NY 10021

Powerful Ideas in Problem Solving and Logo

by Dave Moursund and Sharon Burrowes Yoder

Introduction

The creators of Logo had a vision of young children becoming better problem solvers through immersion in a Logo environment. A large number of researchers into Logo and other computer programming environments have diligently searched for improvements in the overall problem solving skills of students that could be attributed to programming experiences. Such researchers are seeking evidence of **transfer** of problem-solving skills from the computer programming environment to other domains (most specifically, to non-computer domains).

For the most part, these researchers have been disappointed. Most of the research studies suggest that the problem-solving transfer that occurs from students learning to program, be it in Logo, BASIC, or Pascal, is modest at best. Most studies report no significant differences.

Researchers into problem solving would have predicted this lack of transfer. The subject of problem solving and transfer has been studied for many years and there is a substantial body of knowledge in this area. (Dewey, 1910; Fredericksen, 1984; Minsky, 1986; Moursund, 1987; Polya, 1945). Quite a bit is known about when transfer will occur and how to increase transfer. We even know quite a bit about how to teach for transfer.

Logo is an excellent vehicle for creating a rich problem-solving environment. But the Logo environment, all by itself, is not sufficient to produce the desired increases in general problem-solving skills that we would like to have occur. If transfer of problem-solving skills and knowledge is a major goal when having students use Logo, there are many things the teacher can do to enhance transfer. This article is the first in a series that will address how to increase transfer of problem-solving knowledge and skills from a Logo environment.

Transfer of Learning

Learning theorists talk about *near transfer* and *far transfer*. For example, suppose that I learn to tie my left shoe, and I am practicing with brown shoes that have nylon shoe laces. Chances are that the knowledge and skills I gain will transfer to tying the right shoe of a pair of black shoes that have cotton shoe laces. This is near transfer. It is something that most human minds do quite well, quite automatically. The human mind is "wired" to facilitate near transfer.

On the other hand, mastering shoe tying does not readily transfer to tying a bow tie. That is a far transfer. Of course, the terms *near* and *far* are relative. Most of us are familiar with the case of the student who studies and masters metric measurement in a math class, walks down the hall to the next period science class, and claims complete ignorance of any metric knowledge in the science class. Many students find that the transfer from a math class environment to a science class environment is quite difficult—much to the exasperation of teachers. In summary, we know that far transfer does not easily or automatically occur but appropriate practice and instruction can help it. We also know that a number of measures of intelligence are, in essence, measures of transfer. Less gifted students have more trouble making transfers of their problem-solving skills from one environment or domain of discourse into another.

We know how to increase far transfer. Two techniques that will be stressed in this series of articles are:

1. Help students to have a very explicit understanding of the problem solving skills and techniques that they are learning and practicing.

2. Help students to find examples from a wide variety of disciplines where these same problem-solving skills and techniques are applicable. Facilitate students gaining practice in applying their new problem-solving skills and techniques in these disciplines.

Some Things We Know About Problem Solving

We know quite a bit about problem solving. In this section we list a few of the known facts. The reader should be aware that these facts are not 100% guaranteed to hold in all instances. However, there is sufficient research underlying the basic ideas to strongly support curriculum design and instruction based on the ideas. Some examples:

1. Problem solvers who talk about the steps they are taking to solve a problem do better than those who do not describe their work.

   Talking about the problem-solving steps one is taking or contemplating seems to help increase understanding of how these steps relate to each other and to their intended outcome. All teachers recognize that they have learned a great deal through their attempts to explain things to students.

2. How we think about or represent a problem is a better indicator of a problem's difficulty than any quality

intrinsic to the logic of the problem.

Two people, with essentially equivalent backgrounds and experiences, may view a particular problem in entirely different ways. For one the problem may be trivial, while for the other it may seem impossible. This suggests that we should give specific instruction and practice in viewing (thinking about, representing) the same problem in a number of different ways. Much of the work of de Bono (1970) focuses on this idea.

3. Even with well defined problems, people tend to frame small subgoals and may not be able to explain why they did so.

   This is essentially a statement that solving hard problems is hard. It is easy for the human brain to essentially be overwhelmed by a problem. In the absence of careful training as to ways one might proceed in this case, most people tend to select some subgoal and work on it—often with little understanding of why they are selecting the subgoal or how success on it relates to solving the original problem. This suggests that we should give specific instruction to students on ways to get started when a problem seems overwhelming.

4. The typical person has a few basic general problem-solving strategies which they use for dealing with a variety of problem situations.

   This fact suggests that adding even a few problem-solving techniques to students' repetoire may contribute substantially to their ability to solve problems.

5. Precise thinking (processing) is one of the keys to strong problem-solving ability.

   Precise thinking can be emphasized in every discipline. Some disciplines, such as computer programming, tend to place greater emphasis on precise thinking than do other disciplines. Precise thinking and precise representation of one's thinking are, of course, closely related.

6. Experts outside of their domain of expertise do no better than novices. Good problem-solving ability in one area does not automatically carry over to problems in another area.

   Research into problem solving often focuses on *domain-free* and *domain-specific* knowledge. Expertise in a particular field requires a huge amount of domain-specific knowledge. We cannot expect to make students into expert problem solvers in all domains merely by helping them to learn general-purpose problem-solving techniques in a particular domain (such as in computer programming). However, the value of increasing a student's domain-free problem-solving knowledge and skills is self evident.

Powerful Ideas

Careful analysis of the literature on problem solving and the above ideas has led to the identification of a number of *powerful ideas* Here are some powerful ideas that occur in Logo and in a number of other problem-solving domains:

1. Break big problems into a collection of smaller problems.

2. Build on the previous work of yourself and others.

3. Verbalize when problem solving. (For example, it is often quite helpful to explain to someone else what it is that you are doing.)

4. Look for patterns and make use of repetition.

5. Draw a picture, a diagram, a graph, or some other picture-like representation of the problem.

6. One's procedures for solving problems often contain bugs. Learn how to detect and correct bugs.

7. When solving a complex problem by breaking it into smaller problems, it is important that steps used to solve one of the smaller problems not change the other problems or affect steps one is using to solve them. (In Logo, this is closely related to the idea of State Transparency —returning the turtle to its original location and heading at the end of a procedure.)

Criteria for Powerful Ideas to be Included in this Series of Articles

In this series of articles we will examine and illustrate a number of powerful ideas. Ideas to be illustrated must meet three criteria:

1. The idea must be powerful (strongly applicable) in a number of problem-solving domains. However, it need not be universally applicable or equally powerful in all domains.

Powerful Ideas in Problem Solving and Logo — CONTINUED

2. The idea must occur naturally in Logo and be easily illustrated, even to quite young children, in the Logo environment.

3. The idea must be easily illustrated in a number of other problem-solving domains, at a variety of grade levels and in a variety of subject matter areas.

Reader Input Requested

We intend to write a substantial series of articles for the *Logo Exchange*, each focusing on one or more powerful ideas. Reader input is welcome. What are your favorite powerful ideas on problem solving? How do you illustrate them to your students? Please send us some examples.

References

de Bono, E. (1970). *Lateral Thinking: Creativity Step by Step.* New York: Harper and Row Publishers.

Dewey, J. (1910). *How We Think.* Boston: D. D. Heath.

Fredericksen, N. (1984). Implications of Cognitive Theory for Instructional Problem Solving. *Review of Educational Research 54*: 363-407.

Minsky (1986). *The Society of Mind.* New York: Simon and Schuster.

Moursund, D. (1987). *Roles of Computers in Problem Solving: An Independent Study Course.* Eugene, OR: International Council for Computers in Education.

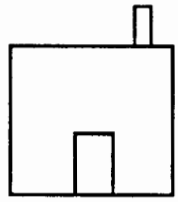
Polya, G. (1945). *How to Solve It.* Princeton, NJ: University Press.

Breaking Big Problems in Small Pieces

A common thread running through the Logo literature is the idea of using procedures and subprocedures. Examples abound showing teachers and students alike how to write procedures and to use them to break drawings into component parts. How many of you have seen an example showing how to write a SQUARE procedures and then use that procedure to write a SPIN.SQUARE procedure? Or, perhaps you've encountered the more common example, showing how to use SQUARE and TRIANGLE procedures to draw a house?

These classic examples seldom go beyond breaking drawings into component polygons in a rather obvious manner. Further, little is made of the need to assist students in looking for ways to subdivide their projects into meaningful chunks. Too often it seems that the teacher simply points out that the house is a triangle and a square and doesn't encourage the student to see it differently—for example, as a *u* shape with a triangle on the top.

Below is a drawing of a not-so-typical house that can easily be done in Logo. Before you read further, take a few moments to think about dividing this simple picture into components. Try to take your thinking out of the typical Logo context. Think about drawing the picture with pencil and paper. Does that give you new insights? Think about the actual programming process. Do you see patterns in what is needed to produce this drawing?

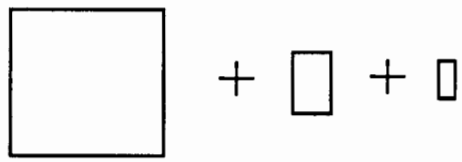


Solution #1:

To solve this problem, you might simply use the turtle to trace the outline of the figure. A key idea here is that you will need to go over some lines more than once (e.g., the bottom of the door and chimney). You might trace the outline of the house clockwise or counter clockwise or you might go around the door and or chimney clockwise or clockwise. If you think about it for a minute, you can see that there are quite a few different solutions that will produce this drawing using only a series of FORWARD's, BACK's, RIGHT's and LEFT's.

Solution #2:

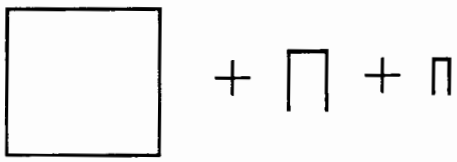
Here the house is viewed as a square with two added rectangles:



Solution #3:

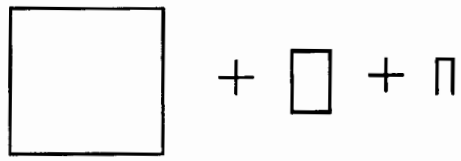
Here the house is viewed as a square with two upside down U-shapes:

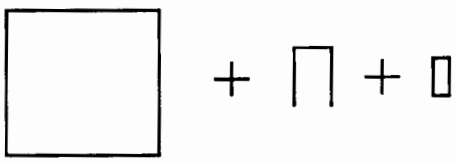
Powerful Ideas in Problem Solving and Logo — CONTINUED



Solution #4:

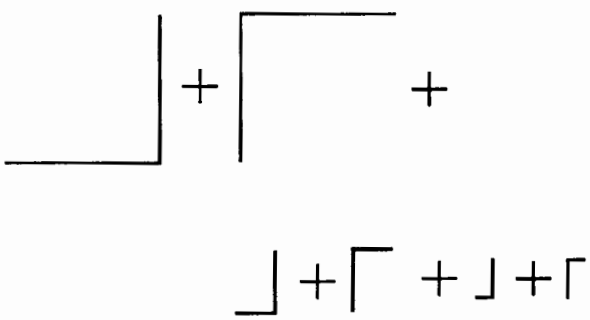
Here we see two combinations of #2 and #3:





Solution #5:

Yet another approach is to view the entire picture as made up of L's. Only one solution using this idea is given below. There are a number of possible combinations of such L's that might be used to create the house.



Did you think of some of the solutions given above? Did you see still other solutions? Were they quite similar to those shown above or were they quite different?

In writing a computer program to solve a problem such as drawing this house, there are two possiblities. First, if you already know how to accomplish the task, you simply do it. That is, if you are given a picture of a square to draw, and you *know* that REPEAT 4 [FORWARD 50 RIGHT 90] draws a square, no further thought is required. However, if the necessary tools are not at your fingertips, then you need to break the problem into parts. In the final analysis, the parts must meet two criteria:

1. Each piece is a problem that you can solve.

2. Doing all the pieces solves the overall problem.

So, in order to draw the house, you needed to break it into pieces (since it is unlikely that you have in mind exactly how to draw a house like the one shown.) If you think about drawing this house *in Logo* then you will likely choose a breakdown that meshes with what you know how to do in Logo. If you are comfortable drawing rectangles, then the square + two rectangle decomposition is the one you are likely to choose to implement. If drawing L shapes has been part of a recent project, then you might choose to decompose the house into a shape for which you have relevant Logo code "at hand."

There are a number of ways that you can practice this idea of decomposing drawings into subparts with your students. You might present simple drawings or designs and discuss with your class different ways to decompose them. You might divide your class into small groups and have each group discuss different approaches and then share their work with the class. Yet another approach is to use student sketches of their project ideas. You can discuss possible decompositions with them individually, in small groups or with the class as a whole. The important idea here is to provide a variety of practice for your students and to make quite explicit that there are many different ways to solve any particular problem. Be sure that you don't discount a student's solution to such a problem because it is not the way you would solve it. Let students discuss the advantages and disadvantages of their solutions, both in and out of the context of using Logo.

The house example discussed above illustrates one of the fundamental issues in teaching problem solving. One can help the student to acquire more and more building blocks—specific problems that they know how to solve. The student will be able to solve more and more problems by merely recognizing that a new problem is one that has been previously encountered and mastered. Or, one can give students lots of practice in breaking big problems into small problems, with the goal of breaking a problem into small problems that one knows how to solve.

Powerful Ideas in Problem Solving and Logo — CONTINUED

How much instructional time and learning effort should be spent on the former task, and how much on the latter? Does an appropriate division of instructional and learning effort vary widely with the student? If so, can we individualize instruction to accomodate these major differences in learning style, and help the student to learn how to learn?

Transfer Examples

As we indicated in the introductory section of this article, teaching a problem solving concept using Logo is only one step in the process of helping students to become better problem solvers. It is necessary to actively teach for transfer of newly learned knowledge and skills in problem solving.

Sometimes the computer teacher is a specialist and does not teach other subjects to the students. Sometimes the classroom teacher is also the Logo teacher. In either case, there are a number of ways to help students to transfer a newly learned technique to other disciplines.

The Logo teacher can point out ways in which an idea learned in Logo applies to another subject. The students can be asked to give examples from other disciplines. Work can be done in large or small groups or individually.

If the person teaching Logo does not teach other subjects, it is especially important that he or she communicate with subject-area teachers the topic being studied so that they can incorporate the new ideas into their teaching. It will be most effective for the learning of students if the connections with Logo are made spontaneously as part of the teaching of another subject. For example, in a math class, the teacher might be discussing how to find the area of a complicated shape by breaking it into subparts. The teacher could include in the discussion that this technique is "like breaking problems into parts using procedures and subprocedures in Logo," and then ask students for some examples. The subject area teacher need not be a Logo expert to assist students in making these connections.

It is very important that students discover for themselves examples where their Logo problem-solving ideas are applicable in other disciplines. The role of the Logo teacher and other teachers is to facilitate this discovery process. The teacher may find it useful to have quite a variety of examples in mind when undertaking this facilitation process. Here are some examples where breaking big problems into smaller problems may be useful.

1. Getting dressed to go to school. This problem can be subdivided in a number of ways. The first consideration might be the weather or the season. Once that is determined, then such matters as color, style, and what matches what need to be examined.

2. Running a series of errands. Presumably the person running the errands knows how to complete each errand on the list. Here the process of breaking into subparts might include grouping the errands by location, ordering them by importance that they be completed, or perhaps grouping them by when would be the best time to do each one.

3. Doing homework. Homework might consist of straight-forward tasks such as "do problems 1-10 on page 23" and more complex tasks such as "write a book report." The student needs to subdivide the larger tasks into "doable" subtasks and then organize the available time in order to complete that work that is due the next day.

4. Cleaning your room. For many a youngster, this is a formidable task, especially as the teen years approach. Subdividing the job into such tasks as clean up the clothes, take care of the records and tapes, organize the books, etc. may make it more manageable. However, if the room is simply too awful, then more subdivision may be necessary: put the dirty clothes in the laundry, put the clean clothes in the drawer, hang up the clothes on the closet floor, and give to your younger brother the outgrown clothes!

5. Cooking a meal. Preparing a meal of more than one dish requires planning. First, the steps to prepare each item must be examined. Then the overall plan for timing the preparation of the meal must be made. The preparation of each item requires a number of steps which might include finding the recipe, getting the ingredients, combining the ingredients in the proper order, and cooking for the required period of time.

6. Learning a new skill in physical education. Often learning to play a new game requires putting together pieces of kinesthetic knowledge into a new "move." For example, most children know how to bounce a ball, but dribbling a basketball requires learning the component parts and putting them together so that the ball can be moved down the court.

7. Decoding a word. When students encounter a new word, they are often told to sound it out. This amounts to breaking the word into subparts that they know.

Powerful Ideas ... — CONTINUED

8. Writing a story. Writing can seem to be an overwhelming task to most students. Drawing a parallel between writing a Logo program and writing a story may relieve some anxieties. They can learn to see that creating the beginning, middle, and end of a story is not unlike creating the parts of a picture in their computer class.

9. Learning a new song. Whether memorizing a song to sing or learning to play a part on an instrument, few students try to tackle the whole job at once. Music is naturally procedural, with phrases bearing a close resemblance to individual Logo procedures. (In fact, if your version of Logo includes music, the writing of music is an excellent way to make using small procedures very natural.)

10. Searching for information in a library. Locating the needed sources in a library can be an overwhelming task without some subdivision. Note that this task requires that you understand the subdivisions of the library, e.g., periodicals, books, microfilm etc., as well as an understanding of how to locate the information within a subidivsion, e.g., card catalogue, computer terminal, searching through stacks, etc.

This list is, of course, by no means exhaustive. No doubt you (and your students) can—and should—think of many more. You might even want to keep a list of new ideas on a bulletin board in your classroom when you are specifically discussing procedures and subprocedures and breaking problems into parts.

David Moursund and Sharon Burrowes Yoder
ICCE
University of Oregon
1787 Agate St.
Eugene, OR 97403-9905

LogoWriter Sound Contest

Walker Junior High School gives a first prize of $50 to the school that sends the best sound procedure. Procedures must be original and use TONE. Entries must include the name of school, grade level, teacher, procedure, and name of sound simulated. Entries will be judged on how closely they resemble the chosen sound.

Send entries to:
Walker Computer Club; c/o Edwina Walsh; Walker Junior High School; 8132 Walker St.; La Palma, CA 90623-2097.

Teaching Tools

"Programming with Style"

by
Glen Bull and Gina Bull

Students in computer science are responsible for meeting coding standards. Part of their grade may be based on how well their programs meet these standards. Why programming standards? Procedures written by professional programmers are frequently reused in other programs. If a procedure can be reused, it is possible to amortize the investment of effort across several projects. Therefore in computer science a great deal of attention is devoted to construction of procedures which are well written.

As you may have guessed, this is the concept which underlies a software tool. The title of this column, "Teaching Tools," stems from this idea. There are other reasons for creating well-written programs. A well-written program is less likely to contain errors. If there are errors in the program, properly constructed procedures are easier to debug.

If a procedure is well written, it may be easier to see patterns in the program. This encourages the "aha!" phenomenon — instances in which the individual suddenly sees a relationship. This is a particularly important part of problem-solving. Well-constructed procedures are easier to modify and change. A well-written program also is easier for someone besides the programmer to understand. This makes it possible for other people to use your procedures. An entire branch of computer science, called "software engineering," is devoted to the study of how to write good programs.

You do not need a degree in computer science to write good programs. A few common-sense ideas will help ensure that your programs will run the first time, every time ... Well, probably not every time. However, the percentage of procedures that do run right away will be higher.

You may be asking whether "software rules" will stifle creativity and exploration? Perhaps this is why very little is found in the Logo literature about ways of writing good programs. Good programming techniques will not interfere with exploration and tinkering. In fact, they will increase the chances of interesting discoveries. Here are four strategies for constructing good Logo programs.

Procedure Length

The single most important strategy is to write lots of short procedures, rather than a single long procedure. Often people

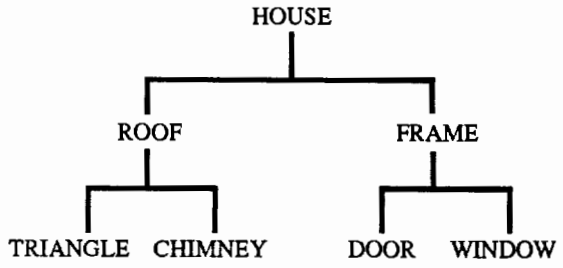
who first learn to program in BASIC write long procedures in Logo too. However, in BASIC it is not possible to create independent procedures—subroutines are the closest equivalent. Logo makes it possible to break a program up into many small parts—using short procedures takes advantage of the strength of the language.

How short should a procedure be? One rule of thumb is that if a procedure is longer than a screen (25 lines), it is too long. Another rule of thumb is that code which is used in more than one place should be written as a subprocedure. For example, if you are creating a house, and have several windows, it is probably a good idea to create a general-purpose WINDOW procedure.

Why are short procedures important? The most important reason is that it is easier to spot mistakes if procedures are short. If a procedure is only seven lines long, there are not many places where the error can be. Of course, it is important to test and debug each subprocedure before using it in a larger procedure.

Using short procedures also encourages good problem-solving techniques. Papert refers to the process of breaking ideas into "mind-sized" chunks. Logo provides an excellent laboratory for experimenting with problem-solving strategies, because it is immediately possible to determine whether a solution works or not. However, providing a child with Logo does not automatically confer these benefits. It is necessary for the teacher to point the way, and suggest effective techniques for overcoming problems. Using short procedures also makes it more likely that some of the procedures will be usable in other programs.

Sometimes it is helpful to construct a procedure tree showing the relationship of the procedures to one another. For example, here is the procedure tree for a house. (Our son asked for a tree house, but our closest approximation was a HOUSE tree!)



The procedure tree makes a useful debugging tool. This procedure tree shows that the HOUSE procedure is three levels deep. At the first level, HOUSE is the master procedure. It is possible to see that HOUSE consists of only two subprocedures: ROOF and FRAME.

```
TO HOUSE
 ROOF
 FRAME
END
```

Of course, it is not necessary to create a procedure tree for a program as short as HOUSE. However, as a program becomes more and more complex, aids such as this make it easier to see how a program is structured.

Use Meaningful Names

Use of meaningful names is the second most important strategy that you can adopt to make better programs. As Papert notes in *Mindstorms*, early versions of BASIC only permitted names which were one or two letters long. As a consequence, people who learn BASIC before using Logo sometimes have a tendency to use names such as X1 and X2. Logo permits a name to be any length. Use of meaningful names takes advantage of the strength of the language. (Some of the more modern versions of BASIC now permit meaningful names as well.)

Of course, it is possible to use any name you want. The computer doesn't care. If you like, you can call a procedure that draws a house RUTABAGA. However, your procedure will be easier to understand if meaningful names are used. Compare the following three procedures that draw the frame of a house.

```
TO FRAME
 WINDOW
 DOOR
 DOORKNOB
END

TO A
 B1
 B2
 B3
END

TO RUTABAGA
 ARDVAARK
 APRICOT
 ENDICOT
END
```

Even though all three procedures do the same thing, the first one is much easier to understand. The second procedure uses numbers and letters instead of names. The third procedure uses names which are not related to the purpose of the subprocedures. Therefore it is difficult to follow what the program does. In a

simple program like HOUSE meaningful names are not as important. As programs grow more complex, it is easier to understand how they function if meaningful names are used.

Meaningful names should be used for procedure names as well as input and variable names. Consider the following Logo procedure that draws an arc.

```
TO ARC :DEGREES :DIAMETER
 REPEAT :DEGREES / 10 [RIGHT 5
      FORWARD :DIAMETER * PI / 36 RIGHT 5]
END

TO ARC2 :X :Y
 REPEAT :X / 10 [RIGHT 5
      FORWARD :Y * PI / 36 RIGHT 5]
END

TO PI
 OUTPUT 3.1416
END
```

In the first procedure, ARC, it is immediately apparent that the first input, :DEGREES, represents the number of degrees in the arc, and that the second input, :DIAMETER, represents its diameter. In the second procedure, ARC2, the user must examine the procedure to determine that :X represents degrees and :Y represents diameter. In this short program, it is relatively easy to identify what the two inputs represent. As programs become more complex, it is more difficult to decipher the purpose of variables.

Use of meaningful procedure names and variable names helps make code self-documenting. It is not necessary to have a separate document which explains that :X is degrees and :Y is diameter if the actual names :DEGREES and :DIAMETER are used instead. This makes it easier for the programmer to follow the logic of the program as a procedure is developed. It also makes it easier for others to understand what inputs to use when they run the program or modify it.

Formatting

Formatting is another area which is typically considered in the construction of programs. In programming languages such as Pascal, attention is usually given to indentation of lines and the physical appearance of procedures. Formatting guidelines of this kind are intended to improve the readability of the code.

Most versions of Logo do not permit indentation, and therefore this is a moot point. However, there are a couple of other factors which can be considered. It is a good idea to group similar commands so that patterns in the program are visible. Consider these two procedures.

```
TO SQUARE1
 FORWARD 50 RIGHT 90 FORWARD 50
 RIGHT 90 FORWARD 50 RIGHT 90 FORWARD 50
 RIGHT 90
END

TO SQUARE2
 FORWARD 50 RIGHT 90
 FORWARD 50 RIGHT 90
 FORWARD 50 RIGHT 90
 FORWARD 50 RIGHT 90
END
```

In SQUARE2 it is evident that there is a pattern in the procedure. It is also possible to discern this in SQUARE1, but it is not as apparent. The pattern leaps out in SQUARE2.

It is also a good idea to limit the length of a line to no more than a couple of commands. After more than one or two commands are placed on a line, the procedure becomes difficult to read.

State Transparency

If a software tool will be used in many different programs, it should ideally leave the programming environment in the same condition in which it found it. This idea is known as *state transparency*. Here are couple of examples.

A programming tool which we frequently use is the procedure OVER. OVER just moves the turtle over.

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

OVER is useful, because it substitutes a single command in place of five. However, the version of OVER shown above could be improved. The difficulty is that OVER puts the turtle's pen down at the end of the procedure. A better version of OVER would record the condition of the turtle's pen at the beginning of the procedure, and return the turtle's pen to the same condition at the end of the procedure.

Teaching Tools — CONTINUED

(PROGRAMMING CHALLENGE: If you would like to write an improved OVER procedure that is state transparent, you can use the command PEN in Apple Logo or the command TURTLESTATE in Terrapin Logo to determine the state of the turtle's pen.)

Here's another example of state transparency. The first procedure, HOUSE1, is not state transparent. It leaves the turtle up on the roof when the house is finished. The second procedure, HOUSE2, is state transparent. It leaves the turtle back on the ground where it started when the house is finished.

```
TO HOUSE1
 SQUARE 50
 FORWARD 50 RIGHT 120
 TRIANGLE 50
END

TO HOUSE2
 SQUARE 50
 FORWARD 50 RIGHT 120
 TRIANGLE 50
 LEFT 120 BACK 50
END
```

If only one house is drawn, there does not appear to be much difference between the two houses. However, if a row of houses is drawn, differences between the two procedures quickly become apparent. Try comparing these two commands. (Clear the screen after each set of houses is drawn.)

```
REPEAT 3 [HOUSE2 OVER 50]

REPEAT 3 [HOUSE1 OVER 50]
```

Were you surprised in the difference in the results you got? The difference is that HOUSE2 is state transparent, while HOUSE1 is not.

Intelligent Guidelines

The most important rule of all is to apply all programming guidelines with intelligence. In the 1970's Dijkstra, one of the leading academicians in the field of computer science, published a paper which noted that GOTO statements can produce unreadable/untestable code. For example, if a GOTO statement is used to jump out of the middle of one procedure into another, it becomes difficult to follow the structure of the program.

In some cases, these observations became translated as "Structured programming means not using GOTO statements." The Logo equivalent of this idea has become "Recursion is better than a loop." Actually, neither is inherently superior; this choice depends on the conditions and purpose. The syntax of Logo does not permit use of the GO command to jump out of one procedure into another, minimizing possible confusion that could result from poor programming.

Blind adherence to any guideline without understanding the underlying reasons is always poor practice. One of our favorite posters is captioned:

GRAVITY.
It's not just a good idea.
IT'S THE LAW!

There are always exceptions to any guideline. If you violate some of the ones we have suggested, the Logo police will not knock on your door and lock you up. However, these ideas will provide some suggestions for development of structured procedures—programming with style!

Further Readings:

If you would like to consult a general purpose reference on programming style, you may want to consider Henry Ledgard's *Professional Software: Volume II: Programming Practice* (Addison-Wesley). The examples are in Pascal, Ada, and C, so not all aspects are applicable to Logo. If you would like to delve into Logo as a programming language the best starting place is a three-volume series by Brian Harvey, *Computer Science Logo Style* (MIT Press).

Glen Bull is a professor in the University of Virginia's Curry School of Education. His CompuServe number is 72477,1637. Gina Bull is a programmer analyst for the University of Virginia Department of Computer Science. By day she works in a Unix environment; by night in a Logo environment.

MathWorlds

by
A. J. (Sandy) Dawson

Gattegno and Papert

The ideas should be born in the students' minds, and the teacher should be the midwife.

Polya

Like many who came to Logo in the late '70s and early '80s, I am trained in mathematics and mathematics education. Unlike very many in North America, my background in these areas has been strongly influenced by the work of Caleb Gattegno. It should not be surprising, then, that I would bring a Gattegno orientation to my Logo work. I see many connections between Gattegno's approach to the teaching and learning of mathematics, and Papert's conception of math worlds. It is these connections which I wish to explore here.

Gattegno, more than Papert, has written about the teaching and learning of mathematics. His approach, known universally as the subordination of teaching to learning, has perhaps had its greatest impact in Europe, particularly Great Britain where Gattegno was the founding chairman of the ATM. His influence is still felt in England as Eric Love notes in his recent review of the Open University text *Routes to Roots of Algebra*, which is an expression of what Love calls "an alternate tradition" [12, p. 49] to the usual approach to mathematics teaching, Gattegno being the inspiration for this alternate tradition.

Elaborations and extensions of Papert's work on Logo, particularly as it applies to mathematics is of course the focus of this column, and everyone here could legitimately be said to be exploring the relationships between Logo and mathematics.

Background

It is worth noting, I think, that both Gattegno and Papert developed their work after a thorough acquaintanceship with Piaget. The fact that Papert's and Gattegno's works contain resonating themes should not be too surprising given this common intellectual heritage.

Gattegno worked very closely with Piaget in the '40s, and indeed was the first translator of Piaget's books into English. Gattegno's early work was influenced by Piaget, but Gattegno went beyond Piaget in terms of how children learn and how mathematics can be taught. Papert also acknowledges his intellectual debt to Piaget. Both men, Gattegno and Papert, came then from similar intellectual traditions with which they have broken. Both were and are deeply involved with the study of how people learn. Both have a very strong conviction about the power of children. Both men advocate exploratory learning. Both men acknowledge the centrality of the teacher to the educational enterprise. Gattegno has written more on the teaching and learning of mathematics and language than has Papert; Papert has published more on children and computers in education than has Gattegno. It is difficult by just reading their published works to gain an understanding of what each stands for because their approach to teaching and learning involves a great deal of doing, contemplating and reflecting. It is no doubt easier to understand the viewpoints from their programs, Logo in the case of Papert, and *Algebricks* and *The Silent Way* in the case of Gattegno. Though all of you have direct experience with Logo, few, I would venture, would have spent a similar amount of time and effort on Gattegno's approaches to the teaching and learning of mathematics. For this reason, I will focus on his work, and then examine how the viewpoints of the two men reinforce each other.

An Alternate Approach

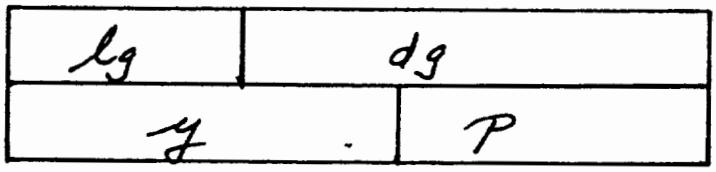
I believe it is true, as Eric Love suggests, that Gattegno's view of mathematics and mathematics teaching is a clear alternative to the typical North American view. But what is this "...alternate tradition which ...has derived much of its inspiration of the thinking of Gattegno"? [12, p. 49] The themes suggested by the Mathematics Education Group at the Open University in their book mentioned earlier are characteristic of Gattegno's approach.

- expressing generality
- rearranging and reversing
- possibilities and constraints
- generalised arithmetic

Let me give an illustration of each of these as a way of explicating Gattegno's orientation to mathematics and the teaching of mathematics.

Expressing generality

Image in your mind, if you will, four Cuisinaire rods—a light green (lg), a dark green (dg), a yellow (y), and a pink rod (p)—arranged in the following fashion:



Math Worlds — CONTINUED

By simply looking at this arrangement of rods, it is possible to see the following relationships among them:

lg + dg <=> y + p lg + dg <=> p + y
dg + lg <=> y + p dg +lg <=> p + y

Now, these observations can in fact be made by kindergarten aged children and coded by them using coloured pencils, because they are recording what they see. They, and certainly older learners, can go a step further and note, again based on seeing, that the following relationships also exist:

lg <=> y + p - dg lg <=> p + y - dg
dg <=> y + p - lg dg <=> p + y - lg

y <=> lg + dg - p y <=> dg + lg - p
p <=> dg + lg - y p <=> lg + dg - y

What learners are doing here is using their powers of seeing, imaging, and evoking to gain awareness of relationships which exist among the four rods. These are not *just* algebraic manipulations in the sense one usually thinks of such work in secondary school mathematics. They are awarenesses whose birth was aided by the visualization of the "dynamics of operations": in this case, the operations of addition and subtraction. The learners have expressed generalities based on their awareness of the relationship among these four coloured pieces of wood, which when translated into standard school curriculum would enable them to deal with a question such as:

Solve the equation a + b = c + d for each of the variables in the equation.

Rearranging and Reversing

Consider for a moment the arithmetic operation of subtraction. North American teachers will universally testify that one of the most difficult things to teach about subtraction is epitomized by the following question:

```
  50000000
- 36748295
```

This is difficult because of the necessity to borrows successively from a whole bunch of zeros.

However, if learners have had opportunities to play with numerals, and to play with the relationships among them, then this question should not pose any difficulty for them, nor should it be necessary to do any borrowing of any kind.

In his computer program, *Mathematics: Visible and Tangible*, Gattegno has the computer present to students a sequence of subtractions designed to *force their awareness* of a certain relationship. You will no doubt recognize it quite easily!

```
17  —>  16  —>  15  —>  14
-10 —>  -9  —>  -8  —>  -7
 7      7       7       7
```

Once learners have seen the effect, or lack thereof depending on how you view it, of this sequence; namely, that diminishing each numeral in the subtraction by one does not alter the difference between the two numerals, then it is a relatively simple matter for them to apply this awareness to that original *difficult* question:

```
  50000000   —>   49999999
- 36748295   —>   -36748294
                   13251705
```

and of course no borrowing was done at all. The numerals were initially re-arranged, and the subtraction was then performed yielding the standard answer. The process could then be reversed—because addition and subtraction are complementary operations—if one wished to check the answer.

Possibilities and Constraints

Picture learners making trains of rods the same length as, say, the yellow rod. They might come up with some of the following trains:

```
r + w + w + w      r + w + r      r + lg    y
w + r + w + w      w + r + r      lg + r
w + w + r + w      r + r + w      p + w
w + w + w + r      w + lg + w     w + p
                   lg + w + w
w + w + w + w + w  w + w + lg
```

As you can see, there is 1 five-car train, 4 four-car trains, 6 three-car trains, 4 two-car trains, and 1 one-car train. If the learners are invited to extend this game to trains of other lengths, they could build a table of the number of trains of various lengths with differing numbers of cars for each of the 10 coloured rods. [You will note that in this game, the train r + lg is considered to be different from the train lg + r.] The chart for the white through dark green rods is given below.

Math Worlds — CONTINUED

Number of trains with *x* cars

	1 car	2 cars	3 cars	4 cars	5 cars	6 cars
white	1					
red	1	1				
light green	1	2	1			
pink	1	3	3	1		
yellow	1	4	6	4	1	
dark green	1	5	10	10	5	1

Here we have a game, playable by most primary aged learners, which is constrained by certain rules and by the length of the rods. It is in some sense a mini-world of all the trains it is possible to make using rods. What are some of the possibilities for awarenesses which might arise from such a world? Consider this scenario. In a class of 15 five-year-olds, is it possible for each child to give a different name for the numeral 5 if the children may only use positive integers and the operation of addition? Think about that for a minute before you read further.

Here are some of the possible names for the numeral 5 if only positive integers and addition are used to generate the names:

2 + 1 + 1 + 1	2 + 1 + 2	2 + 3	5
1 + 2 + 1 + 1	2 + 2 + 1	3 + 2	

As with the trains, the game allows for 2 + 3 to be considered to be a different name than 3 + 2. Do you still think each of the 15 can (cannot) create a different name for 5? Shall we write a few more?

1 + 1 + 2 + 1	1 + 2 + 2	4 + 1
1 + 1 + 1 + 2	1 + 3 + 1	1 + 4
		3 + 1 + 1
1 + 1 + 1 + 1 + 1		1 + 1 + 3

Well, that makes 16 names for 5, folks, so each children could make a different name for 5. Examination of the chart would indicate that there would be 32 different names for 6, 8 different names for 4, and so on. For 10 there would be 512! What richness within such constraints! And, of course, you will recognize that what has been generated here is Pascal's Triangle and hence the possibility for studying binomial expansions also lies hidden within this simple game with trains of rods.

Generalised Arithmetic

Bob Davis, back in his Madison Project days, made the numerals 64 and 28 famous when he wrote about Key's Arithmetic. I want to use those two numerals, but not in a subtraction situation. I wish to analyze the multiplication of them, and in particular to focus on the partial products which are produced.

64	
x 28	
32	(8 x 4)
480	(8 x 60)
80	(20 x 4)
1200	(20 x 60)
1792	

The first partial product is that of the numerals in the ones columns. Then 32 (8 x 4) we have the two products of numerals in the ones column with numerals in the10s column, and finally the product of the two 10s column numerals.

This pattern can be generalized as follows:

$$\begin{array}{r} (a+b) \\ \underline{(c+d)} \\ bd \\ ad \\ bc \\ \underline{ac} \\ ac+bc+ad+bd \end{array}$$

or perhaps of greater interest

$$\begin{array}{r} (a+b) \\ \underline{(a+b)} \\ b^2 \\ 2ab \\ \underline{a^2} \\ a^2+2ab+b^2 \end{array}$$

This illustration, perhaps more than most, reflect Gattegno's description of his approach to the teaching and learning of mathematics as one which is algebraic, because he urges that learners should have their awareness focused on the dynamics of the operations. In this way, arithmetic is algebra if one studies the generalities and not just the particulars.

Math Worlds — CONTINUED

The Subordination of Teaching to Learning [2]

The themes noted above as well as the examples selected to illustrate them, certainly form part of Gattegno's approach to mathematics' education. But they are not all of his approach. Based on his study of learners of all ages, but particularly of babies [4], boys and girls [6], and adolescents [3], Gattegno has concluded that all education is the education one gives to one's awareness. "Only awareness is educable," Gattegno would exclaim, "and that education must be a self-education." That does not mean, however, that teachers sit back and wait for the children in their charge to learn and teach themselves. Far from it. Teachers should endeavor to create situations which gently force students awareness in particular directions, and it is the teacher's job to continually search for ways which will nudge learners along profitable paths to gaining awareness. "Applied to the study of mathematics, awareness is the internal energy which focuses attention on the dynamics of the mind as it studies patterns and relationships. Mathematics is mental activity richly served by imagery." [1,p. 18] Gattegno's approach encourages students to use their powers of visualization and imagery, to not only put words at the service of images, but eventually put images at the service of words. Further, Gattegno advises teachers to stop correcting students. "Errors and mistakes are to be treated non-judgmentally, not ignored or left unrectified, but welcomed as opportunities for new insights to be gained." [1 , p. 18]

Gattegno would encourage children to talk mathematics much as we would encourage young children to talk their mother tongue. He would have children generating hypotheses and testing them against empirical, visual, and logical evidence. He would contend that each new awareness which a learner acquires integrates all which that learner has previously learned, and in so doing subordinates the old knowledge to the new. Awarenesses have a Janus like quality about them! It is not the case, he would argue, that the old accommodates or assimilates the new, but rather that the new integrates the old, and the subordinates the old awarenesses to the new awareness.

Logo and Mathematics: Papert and Gattegno

How does all this apply to the teaching and learning of Logo? You can probably see the connections as well as I can, but let me just point out a few of them.

Expressing Generality—At a simple level, moving from immediate mode to procedural mode is expressing generality. Writing generalized procedures even for such elementary things as BOX where one introduces variables moves the learner very quickly to expressing generality.

Rearranging and Reversing—During a session at the NCTM meeting in Anaheim last April, Johnny Lott demonstrated some Logo programs that examined concepts and ideas about symmetry. Starting from one basic procedure, he had participants explore what would happen if the order of commands in the procedure was reversed or rearranged. What would happen if all right turns were changed to left turns, all forwards to backs, and vice versa? Are the figures one gets mirror images of the original? Always? When and when not?

Possibilities and Constraints—A rather standard early activity for learners just coming to Logo is for each to produce some sort of drawing which covers a sixth of the screen, say, and then for six people to come together and join their drawing together into one program. The possibilities and constraints involved in the exercise are enough to produce many new awarenesses about not just Logo, the language, but also about how to work productively with other learners.

Another illustration of this would be to change the primitives of Logo. (Patrick Mendelsohn gave an example of this last year in Anaheim at his session titled "Computer Programming Objects.") Students were given certain geometric objects as primitives and from these they were to construct other geometrical shapes. [13]

Generalised Arithmetic—I found this theme to be the most difficult to cite an example from Logo, but perhaps you folks can generate some for me. One which did come to mind are the function machines which Paul Goldenberg wrote about in an early MathWorlds column in the *Logo Exchange* magazine. [7] However, it doesn't seem very satisfying to me as an illustration of generalized arithmetic.

Educating Awareness: Forcing Awareness—It seems to me that Uri Leron's arguments about quasi-Piagetian [10] learning bear striking resemblance to Gattegno's idea of forcing awareness. It is not free, raw exploratory work which we wish learners to partake of when doing Logo, at least not for all of the time. We do want them to note and pay attention to what they are doing, to study the patterns and relationships they generate with their procedures as depicted on the screen. And at times we will want to structure situations such that particular awarenesses are brought to the fore; awarenesses such as the total turtle trip, and turtle state, and so on.

Stop Correcting: Self Correction—It is probably even easier with Logo than it is with mathematics per se to stop correcting and to let learners correct their own work. In the mathematics classroom, teachers have to consciously stop correcting students, and they consciously have to try to create situations which enable the learners to correct themselves. With

Math Worlds — CONTINUED

Logo, running procedures and debugging procedures serves both functions. Indeed, it may be the computer's greatest strength in that it allows learners to "mess around," as David Hawkins is fond of saying about science teaching, in an environment where their bugs (errors) are treated nonjudgementally, where they can see the results of their own thoughts as depicted on the screen, and where they can as a result correct their thinking, all without the intervention of anyone else. At times they may wish for that intervention and may indeed seek it out, but then it is on the learners' terms. At times, teachers may want to intervene, but they should make sure they want to intervene in order to help learners learn, and not just so that teachers can teach!

Visualization/playing turtle—To be invited to play turtle is to be invited into the world of visualization and evocation. Words are first used to assist learners to evoke images in their minds which once conceived are then translated to bodily action. Here again the computer, and particularly the computer running a Logo environment, enables learners to carry out this evocation and visualization process rapidly and dynamically. Or the reverse invitation may be issued; namely, to generate words from the images on the screen, and many children have been noted for their creative descriptions of turtle graphic drawings on their monitors.

Guessing and Testing: Conjectures and Refutations—The work of Lakatos [9], derived from that of Popper, has served to legitimize the whole guessing and testing orientation to the teaching and learning of mathematics. Lakatos argued, of course, that the conjecture and refutation process is in fact the manner in which mathematical knowledge grows. While most likely agreeing with this notion, Gattegno would contend, I think, that those guesses, those conjectures, come from watching the mental dynamics of one's mind, of becoming aware of new awarenesses, and testing these out to see if in fact they serve to explicate the situation in question. The Logo culture fosters the development of that guessing and testing atmosphere, and Logo turtle graphics encourages it by providing immediate, graphical feedback about one's guesses.

The New Integrates the Old...Awareness is Educated—"It is without a doubt [learners'] awareness which is being educated. As each new awareness is added to learners' repertoires their old knowledge is re-structured and reorganized so as to serve this new understanding." [1, p.19] In that Johnny Lott NCTM presentation I mentioned earlier, he outlined a Logo procedure dealing with quadrilaterals [Quad :x : y : z] and showed how by focusing on the most general case first, students understanding was heightened. With each restriction of the variables, the particular quadrilateral (rhombus, parallelogram etc.) which might appear on the screen was also restricted. The more generalized the procedure was, the more of the special cases it integrated, and subordinated the special cases to the more general instance.

It should be clear, I think, that Gattegno's approach to the teaching and learning of mathematics fits well with Logo and the manner in which we advocate Logo should be presented to learners. In very many ways, Logo is mathematics, as Uri Leron has argued [11], but Logo fits better with that alternate tradition of mathematics teaching which Gattegno has inspired. Logo teachers would do well to look at Gattegno's approach to several mathematical topics and ideas as possible sources of ideas as to how to use Logo in presenting these mathematical concepts. If this were to happen, then, I contend, we would come closer to Celia Hoyles' plea at Logo 85 that

> Using Logo to promote mathematics investigation must therefore be seen *not just* as a reasonable and efficient pedagogic strategy, but also as a way of challenging the authoritarianism of the deductivist pattern of teaching mathematics by trying to move away from a rather sterile and static formalism to one that is more dynamic and learnable. [8, p. 33]

The above column was given as a presentation at the Third Logo/Mathematics Conference (LME3) held in Montreal, Canada, July 14-18, 1987 and appears in the Proceedings of that gathering.

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

References

Dawson, A. J. (Sandy) (1987). "MathWorlds." *Logo Exchange*, 5(6).

Gattegno, C. (1970). *What We Owe Children: The Subordination of Teaching to Learning*. New York: Outerbridge and Dienstfrey

Gattegno, C. (1971). *The Adolescent and His Will*. New York: Outerbridge and Dienstfrey

Gattegno, C. (1973). *The Universe of Babies: In the Beginning There Were No Words*. New York: Educational Solutions Inc.

Math Worlds — CONTINUED

Gattegno, C. (1974). *The Common Sense of Teaching Mathematics.* New York: Educational Solutions Inc.

Gattegno, C. (1975). *Of Boys and Girls.* New York: Educational Solutions Inc.

Goldenberg, P. (1986). "MathWorlds," Learning to Think Algebraically. *Logo Exchange, 5*(2).

Hoyles, C. (1985, July). Developing a Context for Logo in School Mathematics. *Logo '85 Theoretical Papers.* Boston: MIT.

Lakatos, I. (1972). *Proofs and Refutations.* Boston: Cambridge University Press

Leron, U. (1985). Logo Today: Vision and Reality. *The Computing Teacher.*

Leron, U. (1987). *Logo Exchange, 5*(9).

Love, E. (1986, December). What is Algebra? *Mathematics Teaching.*

Mendelsohn, P. (1986). The Commputer Programming Objects' Project: From Pre-cription to Description of Geometrical Figures. In C. Hoyles, R. Noss & R. Sutherland (Eds.), *Proceedings of the Second International Conference for Logo and Mathematics Education.* London: Institute of Education, University of London.

Polya, G. (1981). *Mathematical Discovery: On understanding, learning, and teaching problem solving* (Combined edition). New York: Wiley.

Correction to Math Worlds:
In a recent column I said that Al Cuoco had developed Scheme. This is incorrect. Scheme is a dialect of LISP developed by Guy Steele and Gerald Sussman. I apologize to Guy and Gerald for this error.

A similar correction came from Brian Harvey

Correction on editorial:
In an earlier editorial, I misrepresented the status of SIGLogo officers. Gary, Ted and Peter will serve as acting officers until NECC '89. The first official slate of officers will be submitted by February 1, 1989 and voted on by April 30, 1989. These new officers will assume their responsibilities at NECC '89 - editor

Logo LinX

"After Before Comes...?"

by Judi Harris

Stanley Kubrick's 1968 space odyssey, "2001," introduced HAL, an almost-perfect computer that appeared to make some fatal mistakes. After "2001" came "2010," during which HAL's mistakes were (of course) discovered to be the result of human misjudgement. But what comes after HAL?

The answer: IBM.

I comes after **H**,
B comes after **A**,
and **M** comes after **L**.

Clever nomenclature, isn't it? It's also an interesting Logo exploration that can help to reinforce notions of alphabetic sequence and phonetic analysis.

Position is (Almost) Everything

What are some other "afterwords"? What spelling patterns (if any) do they follow? In this "afterworld," all vowels become consonants:

A becomes B, E becomes F,
I becomes J, O becomes P,
U becomes V, Y becomes Z.

These consonants become vowels:

Z becomes A, D becomes E,
H becomes I, N becomes O,
T becomes U, X becomes Y.

Many consonants become other consonants.

B becomes C, C becomes D,
F becomes G, J becomes K,
K becomes L, L becomes M,
M becomes N, ...and so on.

Afterimage

Two-letter "afterword" pairs mirror each other with regard to letter type.

OH becomes PI (Vowel-consonant becomes consonant-vowel.)
(V C becomes C V.)
TO becomes UP
(Consonant-vowel becomes vowel-consonant.)
(C V becomes V C.)

Three-letter "afterword" pairs appear in several configurations. Since all vowels become consonants, a V C C word can only become a C V V or a C V C word.

END becomes FOE (V C C becomes C V V.)
ADS becomes BET (V C C becomes C V C.)

Some consonants become other consonants, but a V C V word can only become a C V C word, since all vowels become consonants, and no word can be spelled without a vowel.

AHA becomes BIB (V C V becomes C V C.)

The same reasoning applies to a C V V word, which must become a V C C word. The only other possibility for it to become is a C C C word, which is impossible.

NEE becomes OFF (C V V becomes V C C.)

V V C words that become C C V words are impossible. Can you figure out why? Here's a hint: you may have to make a list of all of the possible V V combinations and their "afterimages."

Aftermath

How many three-letter "afterword" sequences are possible? Which are linguistically feasible? To boggle the mind even further, how many four-letter successions can occur? Here's just one example:

ANTS becomes BOUT (V C C C becomes C V V C.)

Would you predict that there are more two-letter, three-letter, or four-letter "afterwords"? Are five, six, or more-letter "afterwords" possible? If you could generate all possible English "afterwords," would the frequencies (based on length) constitute a mathematical pattern? If so, can you postulate some of its characteristics?

As you may have already guessed, a few simple Logo procedures can serve as helpful tools in this exploration.

Each letter typed in Logo has a corresponding numeric value in ASCII code. The computer will print out this number if the ASCII command is invoked.

PRINT ASCII "A yields the number 65.
PRINT ASCII "B yields the number 66.
PRINT ASCII "C yields the number 67.

Logo LinX — CONTINUED

Conversely,
PRINT CHAR 65 yields A,
PRINT CHAR 66 yields B,
and PRINT CHAR 67 yields C.

Notice that the numbers conveniently increase by one as you type the alphabet in order. We can take advantage of that convention by putting together a procedure that will output the letter immediately following the input :LETTER. Also note that a special result is directed for an input of the letter Z.

```
TO NEXT :LETTER
 IF :LETTER = "Z [OUTPUT "A]
 OUTPUT CHAR (( ASCII :LETTER ) + 1 )
END

PRINT NEXT "A yields B,
PRINT NEXT "B yields C,
PRINT NEXT "Z yields A.
```

Afterworld

The procedure NEXT.WORD uses NEXT to recursively concatenate and output the "afterword" that corresponds to its :INPUT.

```
TO NEXT.WORD :INPUT
  IF EMPTYP :INPUT [OUTPUT :INPUT]
  OUTPUT WORD ( NEXT FIRST :INPUT )
       (NEXT.WORD BUTFIRST :INPUT )
END
```

NOTE: In Terrapin Logo 2.0 and later, substitute EMPTY? for EMPTYP.

```
PRINT NEXT.WORD "AX  yields BY.
PRINT NEXT.WORD "ADD yields BEE.
```

Finally, the superprocedure AFTERWORD prompts the user to type a "beforeword," then prints the corresponding "afterword."

```
TO AFTERWORD
  TYPE [FIRST WORD:]
  PRINT ( SENTENCE [NEXT WORD:] NEXT.WORD
       READWORD )
END
```

NOTE: In Terrapin Logo 2.0 and later, substitute PRINT1 for TYPE, and FIRST REQUEST for READWORD.

The exchange might look like this:

```
FIRST WORD :ITS
NEXT WORD :JUT
```

Afterthought

How many more "afterwords" can you and your students discover? We will publish a compiled list of all entries and participants' names, if you will mail the fruits of your "after-effects" to me by September 1, 1988. Address your output to:

Judi Harris
292 Ruffner Hall
Curry School of Education
University of Virginia
Charlottesville, VA 22903

In addition, you may want to adapt the procedures and ideas presented in this article to view the challenge from a different perspective. What is the purpose of this mission?

"To go where no [one] has gone BEFORE...."
—Cap't. J.T. Kirk

InLXual Challenge

HyperLogo?
by Robs Muir

Programming languages have a way of imposing a particular agenda of ideas on users. Consciously or not, software (and hardware) designers define how a user will approach a particular problem, through what I will term "benign neglect"—an error of omission, rather than commission.

While Logo is not without its critics, I would contend that the designers of Logo have gone to great lengths to include a rich variety of "tools" so that a individual user may not feel so constrained. This is particularly true in the graphics package for which Logo is famous.

The Logo project at MIT helped define a new way of creating graphics on a computer; turtle graphics as introduced by Logo heralded a revolution in accessible graphics. The body-syntonic motions of a turtle give a child (or an adult) immediate access to the pixels phosphorescing on a computer screen.

Contrast this with Cartesian coordinates—the darling of engineers, scientists, and mathematicians. It is easy to understand why turtle graphics are successful in educational setting, especially when one is forced to introduce third-graders to drawing graphics in, say, BASIC.

Unschooled minds can understand and use simple geometric shapes in Logo long before they are capable of understanding those same shapes in languages that enforce the use of Cartesian coordinates. Contrast the following pairs of commands:

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

with

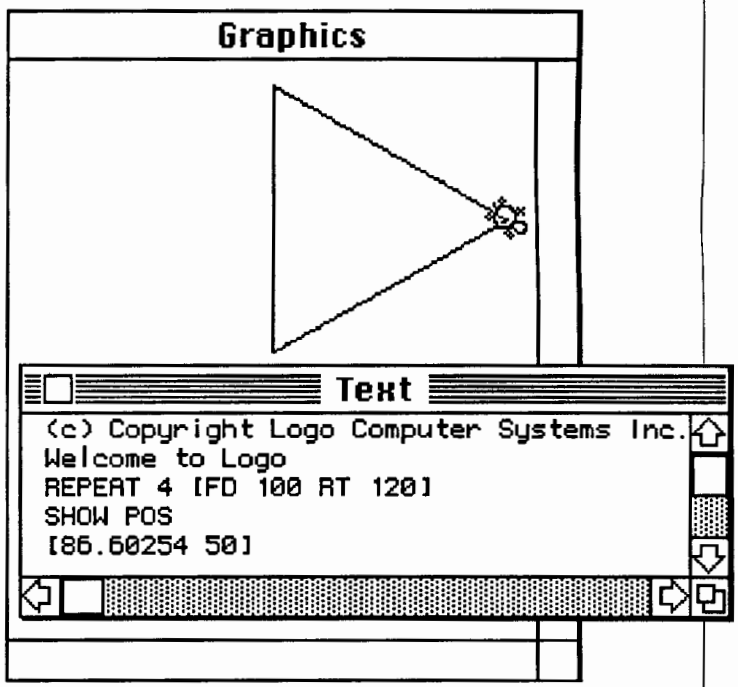
```
HPLOT 0,50:HPLOT 50,50:HPLOT 50,0:HPLOT 0,0
```

Now calculate what the HPLOT equivalent would be if our Logo statement was

```
RIGHT 30 REPEAT 4 [FORWARD 50 RIGHT 90].
```

Can you solve this without resorting to some trig?

Or...consider the following



Why is the POSition reported as [86.60254 50]? Does (100*COS 30) help explain things? Would it help a third-grader?

Fortunately, Logo offers us the choice of using turtle graphics and the implicit coordinate system which is *local* to the turtle, or we may choose to use a cartesian coordinate system (which would require that we use *global* coordinates). As educators, Logo allows us the flexibility to choose the most appropriate graphics tools as we shape environments for ourselves and our students.

The Cards are Stacked

If you have recently returned from an extended vacation in, say, Bhutan, you may not have heard of HyperCard for the Macintosh. HyperCard has been widely touted as a revolutionary software package that promises to become the "programming language for the rest of us." It looks like a paint program, a database management system, and hypertext tool all wrapped in a programming language called HyperTalk. Using a very English-like scripting language, HyperTalk provides a procedural programming environment that can allow a user to write recursive, self-modifying code with power text-handling capabilities. It also includes local and global variables, reasonable mathematics operators, and a semblance of object-oriented programming. Sounds a lot like Logo, huh?

InLXual Challenge — CONTINUED

(As an aside, let me remark that much of the hype about HyperCard is hyperbole. HyperCard will not free us from needing to learn about programming; if anything, HyperCard is a strong testament to the reason for *including* programming instruction in a Computer Education course. I have heard at least one lunatic/fanatic claim that he could teach *anyone* to program in HyperCard in 30 minutes! Sounds just a bit like some of the past promises made by the Logo community...)

After a bit of exploration, I asked some HyperCard experts how (if at all) one could write a set of turtle graphics commands in HyperCard. With the able assistance of David Dunham and Steven Keinle (all via CompuServe), there is now a HyperCard turtle-graphics "stack" that allows you to type things like FORWARD 100 or REPEAT 4 [FORWARD 50 RIGHT 90]. It runs slooooow, but the possibility is there. Since HyperCard includes only Cartesian coordinates in its HyperTalk language, one in forced to use some trigonometry (sine and cosine functions) to calculate the new POSition of the pen after commands like, HOME RIGHT 60 FORWAD 100. The new position should now be [86.60254 50]. (Look familiar?)

HyperLogo–The Challenge

Since we Logophiles forced the HyperTalkers to build turtle-graphics using Cartesian coordinates, it seems a suitable challenge for Logophiles to build turtle-graphics in Logo without the benefit of turtle-graphics commands! Write a set of simple Logo procedures that simulate FORWARD, BACK, RIGHT, and LEFT (call them MY.FD, MY.RT, etc.), the only caveat is that you may not use FORWARD, BACK, RIGHT, or LEFT in your new procedures. You may use things like SETPOS, SETX, SETY, SIN, and COS. You should be able to type statements like

```
REPEAT 4 [MY.FD 50 MY.RT 90] or

REPEAT 360 [MY.FD 1 MY.LT 1]
```

and get expected behaviors worthy of traditional Logos.

Before you begin to think this a redundant Logo project, let me remind you that many computing devices do not have turtle-graphics built in. Several years ago, I wanted to interface an Apple Color Plotter to Apple Logo. To do this, I had to write a "translator" which would convert turtle commands into Cartesian coordinates which were then sent to the plotter. (Incidentally, I'd be happy to send you a listing of this program, if you request it.) This is also a practical application of classroom math—a worthy challenge for some of your students?

PartialLogo on a HyperCard

Turtle Graphics in HyperCard
by Robs Muir

For those of you interested in HyperCard, here is a listing demonstrating how the rudiments of turtle graphics can be implemented in HyperCard using only "native" HyperTalk commands and functions. While the speed could be improved using chunks of code written in C, Pascal, or assembly language—so called XCMDs—this has been intentionally avoided.

David Dunham deserves most of the credit for the following listing. Bill Cook, Fonny Smets, Bradley Poulson, Dan Shafter, and Steven Kienle all helped weave the thread on a CompuServe discussion which culminated in this coding. Any blame for the additional modifications are mine.

Install the script on a BackGround or Card Layer. Turtle commands may be typed in the Message Box (one command at a time). FORWARD, BACK, RIGHT, LEFT, CLEARSCREEN, PENUP, PENDOWN, and SETPOS are currently implemented with this code. For REPEAT and DEFINE, you may wish to download Steven Kienle's LOGO.SIT stack on the Education Library (3) in the APPHYPER forum on CompuServe (GO STACKWARE).

HyperCard gives new meaning to the term *turtle* (read, slooooooow!) After watching *this* turtle crawl, you may have found another reason to continue to use Logo!

```
on openStack
      hide menuBar
      show message box
      set userLevel to 5
      ClearScreen
end openStack

on ClearScreen
      global tX, tY, tHeading, tPen
      get the tool
      choose eraser tool
      doMenu "select all"
      doMenu "clear picture"
      choose it
      put 256 into tX
      put 192 into tY
      put - (pi / 2) into tHeading
      put TRUE into tPen
end ClearScreen
```

PartialLogo — CONTINUED

```
on setPos x, y
      global tX, tY
      put 256 + x into tX
      put 192 - y into tY
end setPos

on penUp
      global tPen
      put FALSE into tPen
end penUp

on PenDown
      global tPen
      put TRUE into tPen
end penDown

on forward dist
      global tX, tY, tHeading, tPen
      put round (tX + cos(tHeading) * dist)
into newX
      put round (tY + sin(tHeading) * dist)
into newY
      if (tPen) then
            get the tool
            choose line tool
            drag from tX, tY, to newX, newY
            choose it
      end if
      put newX into tX
      put newY into tY
end forward

on back disk
      forward -dist
end

on right degrees
      global tHeading
      add (degrees * pi / 180) to tHeading
end right

on left degrees
      right -degrees
end left
```

Robs Muir is a physics and computer science teacher in Claremont, CA and an instructor at the Claremont Graduate School. His CompuServe number is 70357,3403 and his Bitnet address is MUIRR @ CLARGRAD.

Testudinal Testimony

Research on Variables, Algebra, and Logo, Part III: Algebra from Arithmetic

by Douglas H. Clements

In the previous two columns we found that Logo can provide a context for learning some algebraic ideas. However, we also found that students—in and out of the Logo environment possess numerous misconceptions about those ideas.

The Logo/Algebra Link

Soloway (in press) suggests that we may be considering the link between algebra and programming too literally. He asks what it would mean for students who took a programming course to perform better on an algebra test because of their "new and improved" idea of variables. It might be that the programming notion of variable replaced the one they had learned in algebra. But that is unlikely, as early ideas are not replaced (Davis, 1984). It is more likely that a new idea of variable is created in the context of programming. However, if these two ideas co-exist, which would we expect students to use on an algebra test? The answer is, of course, the idea that they learned in the algebra context. Therefore, not much "transfer" should be expected. Soloway's answer—to look for transfer of much more general skills—will be discussed in a future column. However, what else could we do so that the two variable ideas are not committed to solitary imprisonment? The challenge is to build computer microworlds that link Logo and mathematical contexts for variables. This column and the next describe projects that accept this challenge.

Algebra as Generalized Arithmetic

Thompson and Dreyfus (in press) suggest that we view elementary algebra as generalized arithmetic. What is generalized?—the operations of thought normally applied to numbers. Their research indicates that many of these operations of thought have precursors in arithmetic. In addition, the study of arithmetic, particularly integer addition, may be organized so as to incorporate important features of elementary algebra.

Their approach is to focus on the *transformational* aspect of integers, rather than a cardinal aspect. In their Logo microworld, integers are represented by transformations. Students are asked to specify operations so as to develop the ability to conceptualize (a) integers as unary operations; (b) addition of integers as the composition of unary operations; (c) integer expressions as units which themselves are equivalent to some integer; and (d) negation as a unary operation upon integers and integer expressions.

Thompson and Dreyfus met with sixth graders for six weeks,

Testudinal Testimony — CONTINUED

twice weekly, for 40 minutes per meeting. The microworld they explored, called INTEGERS, presents integers as unary operations. An integer is represented as a translation operation acting on positions on a number line. That is, the turtle walks right and left. Entering an integer or integer expression causes the turtle to walk according to specific rules. For example, entering 50 asks the turtle to walk 50 steps in its current direction; -50 causes the turtle to turn around, walk 50 steps, then turn back around. START.AT -70 asks the turtle at the position -70. Vertical lines mark the turtle's beginning and ending positions. The heavy arrow shows the displacement.

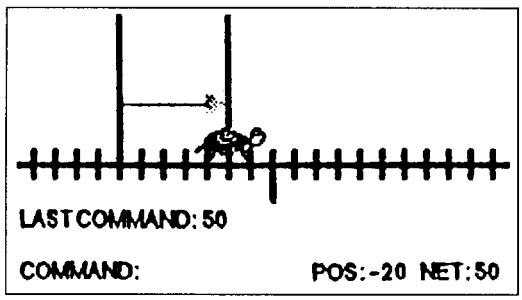


Figure 1

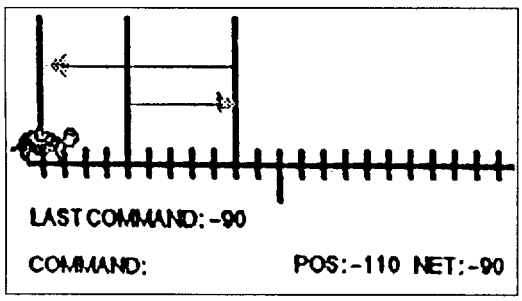


Figure 2

Figures 1 and 2 illustrate first entering 50, then -90. If commands are entered on the same line, they are executed one at a time, with a light arrow showing the net effect (Figure 3). Can you act out the turtle's movement for the fairly complicated problem in Figure 4?

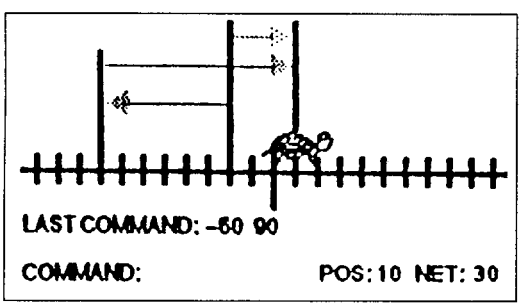


Figure 3

In the two case studies, students were typically asked to predict the result of a command before it was executed, then

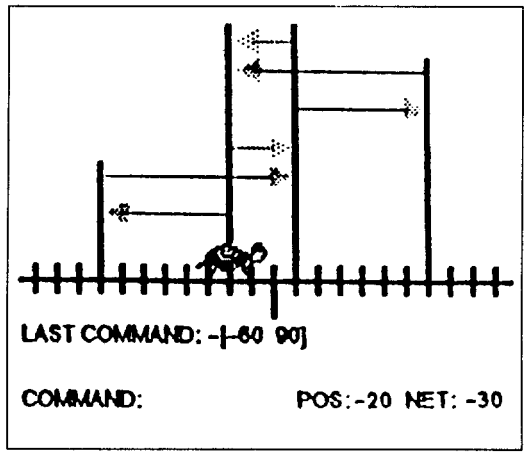


Figure 4

execute the command, then discuss their prediction. The following results emerged regarding the four conceptualizations they investigated.

Integers as unary operations. One of two girls studied extensively soon distinguished between the ideas of position and of change of position. The other confused the two notions for three sessions. She also did not construct direction-displacement as a single conceptual entity. That is, when predicting the effect of entering a negative integer, she might give a displacement but not the direction. Or, she might give the direction as a qualifier: "It will go thirty spaces, and it will go this way."

It took a long time for the girls would specify that—20 was the same as 20. They insisted that though they ended up the same, they were different because "the turtle does a bunch of turn-arounds" for the former but not the latter. Finally, they came to insist that—number is the same as number, "Because it...it turns twice, and then it does...it's facing the way it started out facing [and then it does the number]."

Addition of integers as the composition of unary operations. Specifying the net effect of the composition of two integers was not too difficult for the girls if they were of the same sign. However, it was not until the eighth session that they appeared to reliably identify the net effect of an expression with one positive and one negative number. One girl thought aloud, "...the net effect arrow will be pointing to the number that moved more. Like, if you have umm...let's say seventy and negative fifty.... It'll be pointing to the positive side, because it moved seventy spaces, and seventy moved more than fifty."

Integer expressions as units which themselves are equivalent to some integer; negation as a unary operation upon integers

Testudinal Testimony — CONTINUED

and integer expressions. As expected, the girls originally negated only the first term of a negated expression (e.g., the net of -[20 50] is 30). By the seventh session, they could maintain the scope of the negation, at least in procedural terms (i.e., relating the steps the turtle would perform). But their reliability was low, for this type of reasoning becomes complex with the introduction of multiple negations. With teacher prompts, they began to reason in equivalences of expressions and compositions. For example the teacher asked about the net effect of -50 30. The responded, "negative twenty." He asked about -[-50 30]. They answered, "negative negative twenty." "Do we know what negative negative twenty is?" "Twenty." However, it appeared that only one of the girls was able to use this knowledge spontaneously in solving more complex problems.

Thus, while the girls were still struggling with many of these ideas, they did gain understanding of processes for such tasks as substitutions and the evaluation of expressions. The authors concluded that sixth graders can develop operations of thought in the context of integers that directly parallel those in elementary algebra. Further research would be required to determine whether such experiences affected their later learning of algebra.

The difficulties the girls experienced also calls into question the degree of understanding that students achieve in traditional treatments of integers. Regardless of the method of treatment, it seems that students need a long time to stabilize their conceptualizations. The experiences they are provided for that purpose should emphasize those operations of thought that may generalize to elementary algebra.

Next month's column will discuss a set of microworlds from a newly developed Logo-based pre-algebra course for upper elementary and middle school students.

References

Davis, R. B. (1984). *Learning mathematics: The cognitive science approach to mathematics education.* Norwood, NJ: Ablex Publishing Co.

Soloway, E. (in press). Why kids should learn to program. *Harvard Educational Review.*

Thompson, P. W., & Dreyfus, T. (in press). Integers as transformations. *Journal for Research in Mathematics Education.*

Doug Clements
401 White Hall
Kent State University
Kent, OH 44242

LogoPals

by Barbara Randolph

As an avid learner of tales from far away and long ago, I was recently delighted to receive a legend about "Turtle Island." With great respect, I would like to share a part of this sacred creation story of the Native American Lakota from South Dakota here in the USA.

Water covered the earth everywhere. Four animals were each asked in turn to dive beneath the surface of the sea and bring back mud from the depths below. First, the loon plunged into the water to seek out the mud but with no success. Next, the otter tried and it too returned with nothing. The third animal to attempt the task was the beaver who stayed longer than the other two but also came back with nothing. Finally, the turtle was called because it was strong of heart and fierce in its effort to survive. It stayed so long under the water it was thought it would never return. At last the turtle brought up the prized mud and the new land was made from this patch of mud. This new land was called the "Turtle Continent" or, as some say, "Turtle Island."

Would you like to share your favorite turtle legends or stories from around the world? Send one to me in a letter—I love hearing from new and "old" LogoPals.

It is my pleasure to introduce these new LogoPals who would like to share their ideas and get acquainted with other students of Logo:

Laura Flam (New York, New York, USA): I like to water ski, ride horses and bicycles, and roller and ice skate. I also like to play baseball and run. I would like a penpal from Hollywood or Kentucky.

Tim Jette (Wauwatosa, Wisconsin, USA): I enjoy playing soccer and football. I have learned how to use the Logo editor and how to make shapes with the computer. I am nine years old. I would like a penpal from Port Charlotte, Florida.

Jim Wade (Pennsauken, New Jersey, USA): I am 11 years old. I like wrestling, swimming, and baseball. My favorite color is black. Please send me a penpal.

Andre McKinney (Seattle, Washington, USA): I am eight years old and in third grade. I like to read and fool around with the computer. When I grow up I want to be a scientist. Right now I study animals. Wildcats and birds are my favorites to study.

Jordana Spiro (New York, New York, USA): I have a very big family because I have three sisters and one brother. My mom is from Germany. That is why I would like a penpal from Germany.

Logo Pals CONTINUED

Caroline Horvath (Toledo, Ohio, USA): I am 10 years old. I would like a penpal who lives in France. I like teddy bears and basketball.

Stephanie Lepore (Imperial, Pennsylvania, USA): I enjoy ice skating, gymnastics, choir, band, and swimming. I am in the fifth grade. Please send me a penpal from Japan.

Kristina Kelker (New York, New York, USA): My favorite sports are tennis, ice skating and swimming. I am going to be 10 on March 3. I would like a 10-year-old boy penpal from England.

Yasuko Tate (Seattle, Washington, USA): I am eight years old. I am a third grader. My favorite sport is swimming. One of my favorite hobbies is horseback riding. My favorite activity in Logo is changing the turtle into another shape.

Luigi Massa (Milwaukee, Wisconsin, USA): I am a nine-year-old boy. My birthday is June 15. My hobby is collecting stamps because of their different pictures. I enjoy drawing with Logo. I would like my penpal from Italy.

Jill McDonough (Orchard Park, New York, USA): I like Logo. I also like to ski, swim, play soccer, and track. I like cats. I am in the fourth grade. I am 10 years old. I want a LogoPal from Hawaii or Mexico.

Ben Ruder (Seattle, Washington, USA): I am nine years old and the second oldest in my class. I am in the third grade. My favorite sport is basketball. I have a brother and sister who are both four years old. My favorite Logo activity is learning new commands.

Would your students like to become LogoPals? Then teachers, please share these ads with your students. They can request one of these boys or girls for penpals or they can be matched with other students in our network. Have them write a letter which includes their age and grade, what their hobbies and interests are, and what they enjoy about Logo. A number of student "ads" will be printed in our LogoPals column.

Students in the USA also need to send a self-addressed stamped envelope with their letter. Those outside the USA should enclose international postal coupons (purchased at the post office) for a one-ounce or 28-gram reply.

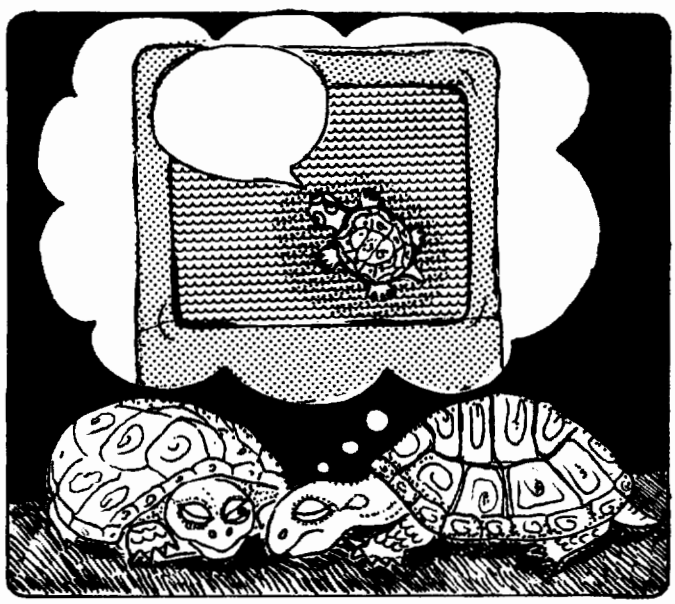
Write to: LogoPals
c/o Barbara Randolph
1455 E. 56th St.
Chicago, IL 60637
USA

Barbara Randolph is a librarian and instructional media center teacher in the Chicago Public Schools.

The Adventures of Jacques and Elsie

drawings by
Linda Sherman

Due to delays in mailing, we're continuing to rely on the questionable judgement of Jacques and Elsie for captions until further notice.



"This must be a nightmare. Jacques promised me we'd never live in a condo."

Send your caption to:

Linda Sherman
Rt. 1 Box 292-A
Shipman, VA 22971

Linda Sherman is a freelance author and artist living in Shipman, VA., with her husband and two-year old son.

Pen Points: Logo Book Reviews with ASTROLUG

by Barbara J. Putnam

Beginning With Logo (Terrapin Version), by Steve Tipps and Glen Bull, 1987, published by Prentice-Hall, Inc. Englewood Cliffs, NJ 07632.

"Drawing a triangle is too trivial a task to be the goal of learning Logo. Learning to describe how to draw a triangle, to recognize patterns, to break a large problem into pieces, to control the computer for personally important and worthwhile tasks—these are the goals of Logo."

Beginning With Logo is a 274-page book for the adult who knows nothing about Logo. As an "intermediate logophile," I enjoyed both the style and the content of this comprehensively written text. I never understood the relationship between THING and : (dots) until now. Reading the book made me eager to try some of the ideas. I highly recommend this book as a preservice or inservice course text for teachers because it thoroughly introduces so many important concepts.

Each chapter has five subdivisions: Concepts and Commands introduces the reader to the programming commands that are needed for that chapter. How Things Work covers technical skills. The Logo Laboratory contains exercises and explorations. Teaching suggestions and project ideas can be found in the Classroom section, and at the end of each chapter is a helpful summary of new commands.

Concepts and Commands sequentially progresses from *turtling*, REPEAT patterns, procedures, and simple text, through variables, recursion, interactive programs, words/lists, and finally a few tools. A large amount of information is covered in an interesting way. The authors suggest that the material in chapters 1 through 4 are appropriate for students up to Grade 4. I would recommend also covering the information on file managing from the How Things Work section in chapter 5.

How Things Work includes basic disk operations, printing, workspace concepts, editing, file management, planning, documenting, and debugging. File managing can be a major stumbling block with beginners. The topic is split into two discussions and handled very effectively. Programming tips such as the following, make the book even more helpful to beginners:

- Never allow a procedure to be longer than one screen.
- Whenever possible, make procedures state transparent.
- Have one procedure called START that contains both documentation and the master procedures for every file.
- Make an ordered printout of all procedures to aid in future changes.

Every chapter has a Logo Laboratory section containing activities that go along with the concepts discussed. There are 10 or more exercises for each chapter. Some I particularly enjoyed were: playing with zigzags, predicting REPEAT outcomes, and making moving messages. There is a list of questions titled Checkup to help you keep track of your progress.

The In the Classroom section contains suggestions, discussion topics, and group project ideas for class instruction. Off-computer activities include body direction games for young students, pattern finding activities, creating word pictures, playing with random numbers, measuring and comparing diameters and circumferences, writing interactive stories, exploring the concepts of variables, data collection, and graphing. Older students can play with plural rules, create madlibs, and formulate quizzes for each other.

Throughout the book, differences between Terrapin versions 1, 2 and 3 are pointed out. The Tools chapter at the end of the book is especially appropriate for users of version 1. There is room for expansion in this chapter. For example, a tool to randomly pick an item from a list and then eliminate that item from the list would be interesting and useful. A discussion of microworlds would also have been an appropriate addition to the book.

Beginning Logo users will find typographical errors extremely frustrating. On the other hand, it can be fun for you and a good learning experience for your students to find the typos on the following pages: 83, 87, 97, 110, 132, 139, 170, 178, 199, 249, 250, 256, 261, and 261. (Also on page 270 the *Logo Exchange* publisher's name and address is wrong.)

If you are looking for ready-made lesson plans and worksheets, this is not the book for you. Although there are many suggested activities, the actual classroom implementation is left to the discretion of the individual teacher. Even if you already know Logo, *Beginning With Logo* can give you some new ideas and some very pleasant reading. In fact, you may find yourself hoping the authors decide to write a sequel. I give *Beginning With Logo* an ASTROLUG thumbs up!

Barb Putnam is a physical education teacher at the Lake George Elementary School, Lake George, NY12845. She has taught Logo to educators for the Greater Capital Region Teacher Center.

Global Logo Comments

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

Logo Exchange Continental Editors

From the "where have we heard that one before" file:

> "We are not using Logo in our school district because we are concentrating on the secondary schools which need a real language like BASIC. When we start putting computers into the elementary schools then we will look at Logo."

This is exactly the message I got from education department officials upon my arrival here in the Virgin Islands. So another selling job for Logo seems in order. Stay tuned. Are there any Logo using teachers out there who want to come to the Virgin Islands and give me a hand?

This month's column highlights Jamaica and Japan. The Jamaica piece was submitted by Edna O. Schack and Markham B. Schack who are professors of educational computing and arithmetic teaching at Morehead State University in Kentucky. They describe a service project sponsored by the University of Texas. The Japan article was submitted by our Asian correspondent, Hillel Weintraub, as his last contribution as our Asian editor. Hillel is on his way to Harvard & MIT where he will be working on a Ph.D. We still hope to hear from him regularly as our "roving international reporter."

Teaching and Learning Logo in Jamaica, West Indies: A Cross-Cultural Experience

by Edna O. Schack and Markham B. Schack

What is it like to teach Logo to students and teachers in a Third World country where many are enthusiastic and self-motivated learners, despite severely overcrowded and poorly equipped classrooms?

We, Drs. Mark and Edna Schack, recently experienced this and more in the West Indies nation of Jamaica. Our work began in January, 1985 when we first participated in a University of Texas sponsored service project. Our main objective was to introduce the faculty and students of Ruseas High School in the rural town of Lucea to their new computer through Logo. The NCR computer had been only recently donated to the school by a wealthy Jamaican merchant. We chose Logo because we believed the faculty and students would be highly motivated by the screen graphics, which they could quickly learn to program. Furthermore, one of the developers of *"TLC Logo,"* John Allen of the LISP Company, was happy to donate the Logo disk and manuals to the high school.

We worked individually with teachers for several days before conducting a seminar with about 25 of the more interested faculty. The most fun, however, was teaching the students in the computer science class. After reading and studying about computers for five months, they were hungry to get their hands on the keyboard. We do not believe any of them had programmed a real computer before this but within hours, they were writing and running their own Logo programs on the NCR. They were thrilled, and so were we.

In January 1988 we returned to Lucea, Jamaica to accompany our old Apple II+ to its new home at Ruseas High School. Before the Apple II+ was completely unpacked, one of the students had his fingers on the keyboard and his mind on programming. Imagine our excitement when several more students gathered, all with a special gleam in their eyes and notebooks full of Logo procedures! These were not the students we taught three years ago, but new students who had learned about Logo from them. Even more rewarding to us was that they had all surpassed the point in Logo we reached three years ago.

These students were eager to advance their understanding of Logo and computers. Their computer science teacher had just recently left for school in the United States, so the students would soon be in charge of their own learning. They were a pleasure to teach. As we shared ideas with them, they shared ideas for

Global Logo Comments — CONTINUED

learning projects their class could undertake.

"We can translate our TLC procedures to Terrapin procedures."

"We can have challenges. One team can create a procedure and another can see if they can reproduce it, maybe with fewer commands."

Learning with Logo had truly become a discovery experience for these enthusiastic students at Ruseas High School. Naturally, we were exhilarated by the opportunity we had given them, but certainly we learned even more than we taught. We hope to return to Jamaica to continue to share in this cross-cultural experience.

Asian Logo News

by Hillel Weintraub

It has been a few months since we've given any report on Logo activities in Asia. This is because we are in the process of changing field editors and trying to reestablish contacts with those who have given us reports before. We are naturally more aware of Logo activities in Japan than in other countries because of proximity and personal involvement. This month we will bring you up-to-date on a major addition to the Japanese Logo scene.

The latest activity here has been the development of *Logo Writer* with Japanese language capabilities. Logo Japan, a subsidiary of LCSI, has been working on this for the past two years and at the beginning of 1988 announced their product. Seymour Papert was in Japan for a series of workshops given to Japanese teachers who had already had experience with using Logo in their schools.

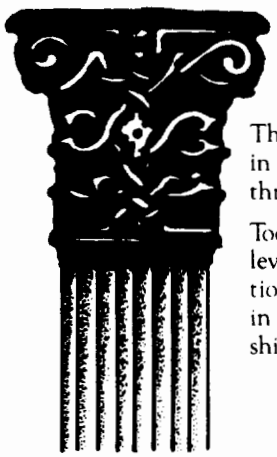
Papert, who has worked closely with the Japanese design team throughout, is aware of the importance of teacher training and teacher support—even more so in Japan than in North America and Europe—because most teachers have little experience with exploratory learning, either as students themselves or as teachers.

From this experience, Logo Japan may see the necessity of giving much of their attention to developing additional classroom and teacher training materials and workshops, so that teachers can be encouraged to support each other in this difficult transition stage that Japanese education is undergoing.

One of the teachers who attended the workshop was the principal of an elementary school in Nishinomiya, about five hours from Tokyo on the Shinkansen, Japan's bullet train. The principal invited Professor Papert to visit the school and observe elementary math education both with Logo and without. A few days later, Professor Papert was in Nishinomiya, getting a chance to see Japanese math education first hand. The teachers who saw Logo and *LogoWriter* in action were very impressed, though many had difficulty relating it to their present concepts of teaching and learning.

Still, it's clear that Professor Papert's and Logo Japan's efforts in creating *LogoWriter* in Japanese are a significant addition to that of other Logo pioneer educators and companies in Japan. It's also clear that no meaningful changes will come quickly. It may even be that when their educational computing curriculum is introduced in the early and mid-90s, the national department of education, which controls and directs education matters, will choose a language other than Logo for the schools to use. It is equally clear that this language will not be BASIC, and whatever the language is called, it will be influenced by the Logo philosophy and style of learning, though, of course, to what degree—well, that is not yet clear.

ICCE



About ICCE

The International Council for Computers in Education was founded by Dr. David Moursund in 1979 as an organization that would foster appropriate instructional use of computers throughout the world.

Today ICCE is the largest professional organization for computer educators at the precollege level. It is nonprofit, supported by 14,000 individual members and more than 50 organizations of computer-using educators worldwide. These organizations are statewide or regionwide in scope, averaging 500 members each. Approximately 84% of ICCE's individual membership is in the United States, 12% is in Canada, and the remainder is scattered around the globe.

About *The Computing Teacher*

ICCE publishes *The Computing Teacher* journal. *The Computing Teacher* provides accurate, responsible, and innovative information for educators, administrators, computer coordinators, and teacher educators. The journal addresses both beginning and advanced computer educators through feature articles, columns, software reviews, and new product and conference listings. Contributors to *The Computing Teacher* are leaders in their fields, consistently providing the latest information in computer education and applications.

Publications, Special Interest Groups

In addition to *The Computing Teacher*, ICCE provides a number of publications to computer-using educators. ICCE's Special Interest Groups provide in-depth information for computer coordinators, teacher educators, computer science educators, and Logo-using educators. *C.A.L.L. Digest* is published nine times per year for ESL teachers. ICCE committees address a variety of ethical and practical issues important to the computer-educating community.

Independent Study Courses

ICCE offers graduate-level independent study courses, designed to provide staff development and leadership. These courses have been approved by the College of Education at the University of Oregon and carry graduate credit from the Oregon State System of Higher Education. Participants correspond with instructors by mail.

Write for information and a free catalog today!

ICCE • University of Oregon • 1787 Agate St. • Eugene, OR 97403 • Ph: 503/686-4414