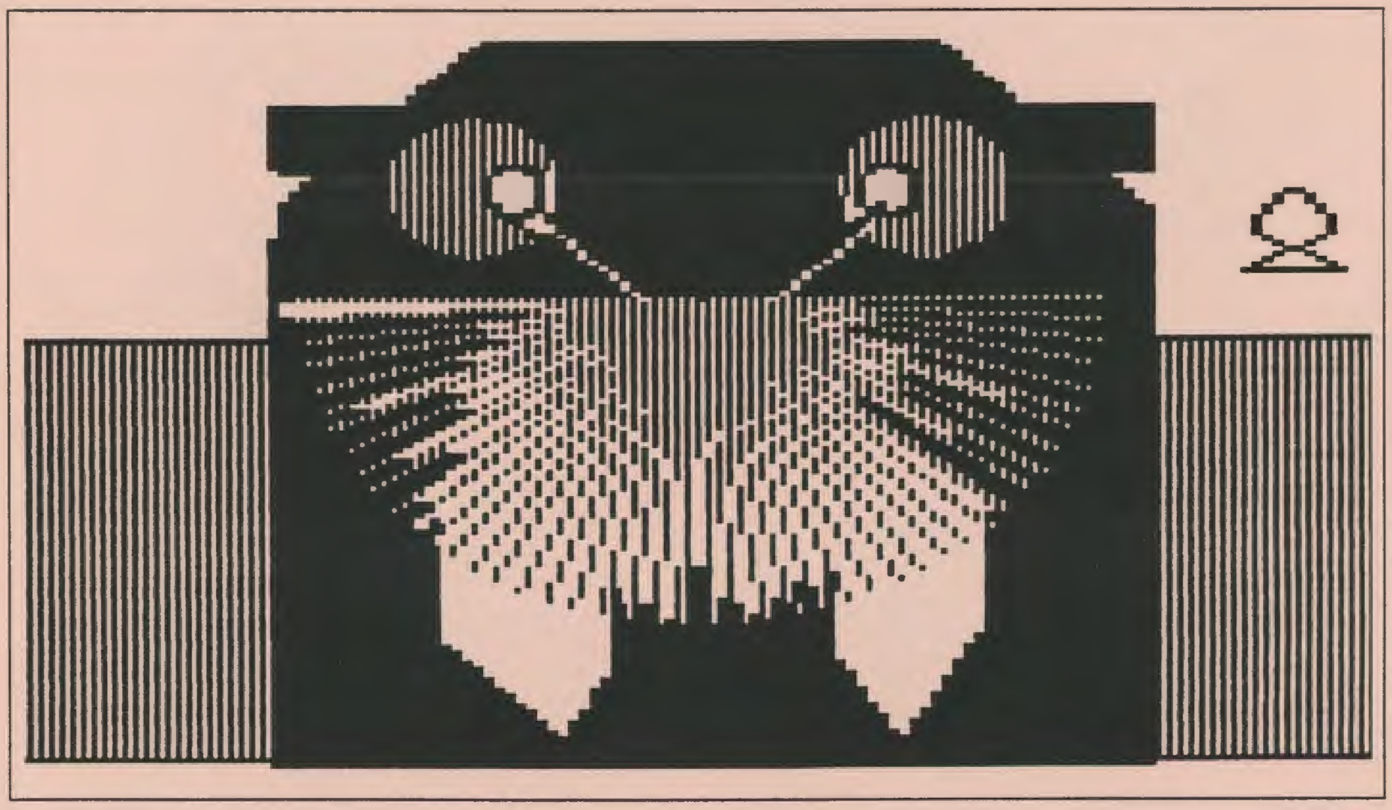
Journal of the ISTE Special Interest Group for Logo-Using Educators

LOGO EXCHANGE

February 1990

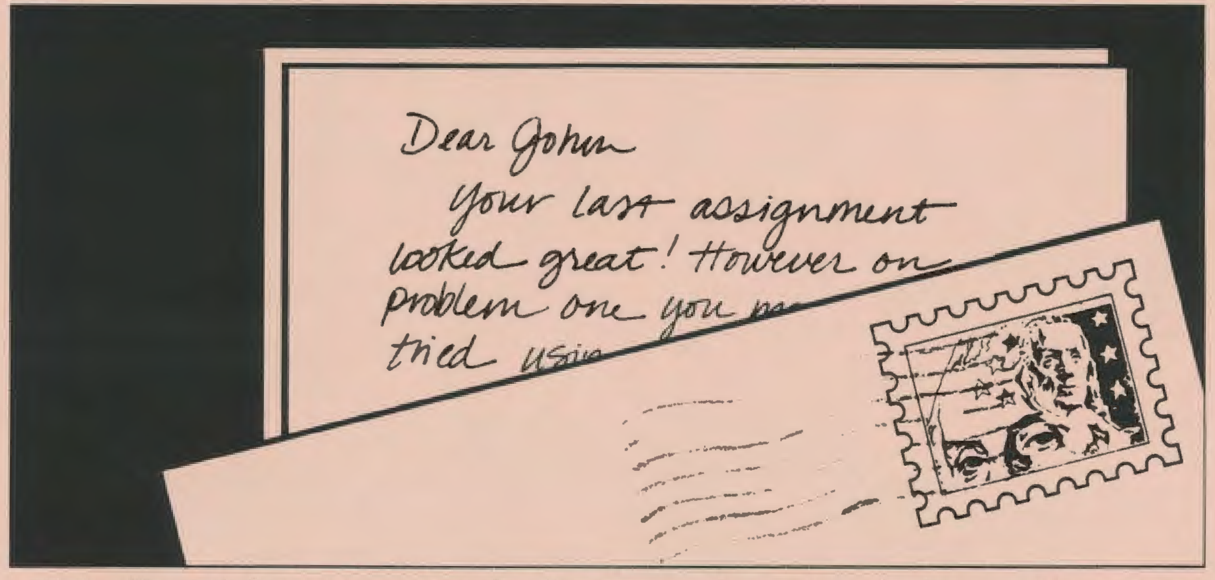
Volume 8 Number 6



International Society for Technology in Education

ISTE Publications

Finally, a long distance relationship that won't break your heart.


Dear John
Your last assignment looked great! However on problem one you ma
tried usin

ISTE offers eight *Independent Study* courses that get to the heart of learning.

Each course thoroughly covers the title material and is designed to provide staff development and leadership training. You correspond directly with the course's instructor by mail, and can receive graduate credit through the Oregon State System of Higher Education.

Classes offered this year are:

- Introduction to Logo for Educators (available for LogoWriter or Logo PLUS)
- Fundamentals of Computers in Education
- Long Range Planning for Computers in Schools
- Computers in Mathematics Education
- Computers and Problem Solving
- Introduction to AppleWorks for Educators
- Computers in Composition
- Effective Inservice for Instructional Use of Computers in Education

Register for classes independently or with a group. Districts enrolling six or more teachers receive a fee reduction for each person enrolled.

Courses range from $222 to $336 for 3-4 quarter-hours of graduate credit. You have one year to complete your course.

Start a great long distance relationship today with an ISTE *Independent Study* Course.

Request an *Independent Study* course brochure. Write or call:

ISTE, Unviersity of Oregon,
1787 Agate St., Eugene, OR 97403-9905
ph. 503/686-4414

LOGO EXCHANGE

Volume 8 Number 6 Journal of the ISTE Special Interest Group for Logo-Using Educators February 1990

Advertising space in each issue of *Logo Exchange* is limited. Please contact the Advertising Mgr. for availability and details.

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

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

Contents

ISTE Membership

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

SIGLogo Membership (includes *The Logo Exchange*)

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

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

From the Editor

Evaluation and Technology

During the last term, I taught a course entitled "Logo for Educators" here at the University of Oregon. Most of the students in this class were graduate students planning to specialize in the area of Computer in Education. The class included students with a wide range of backgrounds, from computer novices to those with undergraduate computer science majors. We had people with very little teaching experience and some with a lot of years in the classroom We had students from the U.S. as well as Taiwan, Malaysia, Korea, and Australia.

Because there is seldom enough class time to cover all of the important issues in Logo education and because I want graduate students who complete such a course to get "beyond turtle graphics," the programming part of the class is often presented in a lecture/question-answer format using a single computer system with a large display screen. Students then do their programming in the labs here at the university as they have time in their own schedules. As is often the case, about half way through the course, students began to complain that my teaching was not "Logo-like." They they wanted lab time in class; that they wanted more interaction with their classmates and a "Logo teacher;" they wanted to experience a "Logo environment."

This is always a point of frustration for both me and my students. I don't have enough time to cover the essential programming concepts as well as important issues such as Logo research, problem solving, and assessment. My students need the experience of a Logo environment. No compromise ever seems satisfactory for all of us. However, I'm always glad when my students complain – it means that they have really begun to understand that Logo is more than a just programming language.

As the end of the term approached, students began to ask about the final exam. (Yes, I do give a final exam in a Logo class.) Part of the exam is short essay questions asking students to synthesize the wide variety of readings that we have done during the term. Part of the exam is Logo programming. Students are allowed to have a "cheat sheet" so that they can look up commands or syntactical questions during the exam. In addition, most of the programming questions are rather open ended, e.g., "Write a procedure that will stamp three different shapes on the screen." The Logo programming questions closely parallel various exercises that they have done during the term. The goal is simply to have the students realize how far they have come in their programming ability since the first few weeks of the course.

In the past students have complained that a traditional final exam was unfair. After all, they wrote their papers on word processors and did their programming "hands on" and then I asked them to perform on paper. This year I had access to enough machines of enough variety to plan for a "hands on" final – something I had frequently done with high school students.

It was particularly interesting to listen to the reactions of even the most sophisticated computer users as they handed me a disk instead of the usual sheaf of papers when they finished their work. Many clearly felt uneasy about leaving their fate to electronic media. Many students commented on what an interesting experience taking an exam on the computer had been.

It struck me as particularly strange that in this computer age even students who are sophisticated computer users and are taking Computers in Education courses have never taken an exam on the computer. For me this raises all sorts of questions regarding evaluation in the information age.

As I have mentioned before in these pages, it is certainly possible to produce products that go beyond desktop publishing with the newer versions of Logo. Moving shapes, changing colors and sound just can't be printed! How do these student products fit into our standard evaluation schemes? And what of the whole area of Hypermedia? (See *Logo &Company* in this issue of *LX*.) How are we going to deal with the evaluation of non-linear products such as those produced using HyperCard? And what of writing? If we encourage children to write with word processors and then they are tested with paper and pencil, how valid are our results? What of standardize tests? How will ETS react to this changing technology?

So, the next time you do an evaluation of any kind in your classroom, perhaps you should ask yourself how you might use the emerging technology to accomplish your goals in a more appropriate manner. I must admit that my recent experience with a "hands-on" exam is going to make me rethink evaluation in all of my classes. How about you?

Sharon Yoder
SIGLogo/ISTE
1787 Agate Street
Eugene, OR 97403
CIS: 73007,1645
BITNET: YODER@OREGON

Monthly Musings

Put It Out
by Tom Lough

"Excuse me, but could you please explain how OUTPUT works?"

I remember asking this question a few years ago, when I was first trying to understand Logo. But I got to thinking about it again recently when someone asked me about it.

In general, procedures can act either like commands or operations (operations are sometimes called reporters). Primitives (or primitive procedures) which are commands include FORWARD, RIGHT, and SHOWTURTLE. Operations or reporters include HEADING, XCOR, and ITEM. Note that the reporters provide a result for use. Note that the command OUTPUT can be used in a procedure to make a reporter, a difficult idea when you first encounter it!

Let's look at a simple example. Here is a procedure which might be part of a project using only words and not numbers to produce the desired results.

```
TO SIXTY
OUTPUT 60
END
```

The intention is to have SIXTY report the value 60. However, running the SIXTY procedure produces the message.

```
SIXTY
I DON'T KNOW WHAT TO DO WITH 60
```

(Or, in Terrapin Logo or Logo PLUS, the message "RESULT: 60" appears.)

OUTPUT simply sends out something without regard to where it is going or what is to be done with it. In order to use the results of an OUTPUT, there must be some preparation. In particular, there must be a place ready for the result of OUTPUT. For example, in

```
FORWARD SIXTY
```

the FORWARD command requires one input and has a place reserved for that. Here, the OUTPUT value of 60 goes there, and is used by the FORWARD.

Sometimes OUTPUT can be used in a chain of information passing. Here is a little more complex example of this.

```
TO HELPER
OUTPUT PICK STUDENTS
END

TO STUDENTS
OUTPUT [CHARLES ALICE JAMES JENNIFER
   THOMAS EVELYN]
END

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

Just for fun, let's run the HELPER procedure.

```
HELPER
I DON'T KNOW WHAT TO DO WITH ALICE
```

Now the error message has an additional meaning! Sometimes you may not know what to do with Alice, either! (This brings up an idea for a later column. What interesting Logo error messages can you generate? Send me your ideas and I will print the most interesting ones. Thanks.) Now let's run HELPER properly.

```
PRINT HELPER
THOMAS
```

That's a little better! So while Thomas is washing the chalkboards, let's focus on the chain of communications among the procedures. PRINT needs one input and holds a place for it. HELPER reports the result of the selection of a student at random provided by PICK STUDENTS. PICK needs one input and reports the value of an item selected at random from its input. STUDENTS reports the list of student names; here, the list is provided as input to PICK. All along the way, OUTPUTs pass information from one procedure to the next.

When I teach, I sometimes give out information on my own initiative. And, sometimes, my students "do not know what to do with" the information I put out to them. Just as when I use OUTPUT in programming with Logo, I need to make sure there is a place for the information I am providing. One way to do this is to wait until a student asks me about something. When that happens, I know that the student has a place for the information I provide and plans for what to do with it. It reminds me of the way FORWARD SIXTY works.

I'm going to do some more thinking about OUTPUT as it might apply to teaching. Won't you join me?

FORWARD ONE.HUNDRED!

Tom Lough, Founding Editor
PO Box 394
Simsbury, CT 06070

Logo Ideas

Just For Fun — Turtle Dances
by Eadie Adamson

Last year I began working with a group of fifth grade students on motion projects with LogoWriter. I had discovered that other groups of students had a great deal of difficulty controlling four turtles in a complex game. I decided to ask the new group to work with multiple turtles in a different way before they launched into their projects. I wanted them to focus on developing some understanding of how to control the turtles before trying to write games or animation. The germ of the idea came from a few pages in Sharon Yoder's *Introduction to Logo Using LogoWriter.*

We began by writing a "dance" for a single turtle. There were a few rules and a few basic ideas to follow:

•The turtle dance should begin and end at **home**, with a heading of 0. (In other words, starting and ending position and heading were to be the same.)

•The "choreographer" should limit the turtle to 25 steps from home in any direction when possible.

•The "choreographer" should experiment with ideas of symmetry and asymmetry in motion: identical movements performed to the left and to the right, with an occasional "passage" not being symmetrical.

•The "steps" created for the turtle should be developed into little procedures which could be combined and recombined at will.

•The final "dance" should be activated by a single command, such as "dance."

We discussed briefly how one might begin such a project. After a few initial experiments we worked out together using our large screen demonstration monitor, it was clear that, in addition to the requirement to begin and end at home, the turtle must begin with its pen up. Our first group experiments involved creating the germ of a "dance" using procedures for squares and circles. The boys quickly realized that there were many geometric ideas which could be appropriated in this fashion. Turtle "spins" were created:

```
to spin
repeat 36 [right 10]
end
```

```
to spin.left
repeat 36 [left 10]
end
```

The boys experimented with spinning while moving:

```
to spinmove
repeat 36 [forward 5 right 10]
end
```

Well... you get the idea, I trust.

Once a simple dance is created for one turtle, it is easy to introduce four turtles. There is already something interesting programmed for each of the four turtles to do, and if each "step" has been programmed as a separate procedure, there is a great variety of movement with which to experiment.

Talking to the Turtles
Begin with the commands:

```
tell all
st
dance
```

It is now possible to create a four-turtle ballet (to be correct, a four-person ballet is a *pas de quatre*). All four turtles "listen" to the command **tell all**; single turtles "listen" when their name is called, as in **tell 1**, while the other turtles wait; using **each**, turtles can perform "steps" in turn.

Put simply, talking to turtles involved:

tell all activates all four turtles
(or **tell [0 1 2 3]** which has the same result)

tell 1 (or 2 or 3 or 0) causes that turtle alone to "listen"

each follows a list of turtles and causes those turtles to take turns performing a task, expressed like this:

```
tell all
each  [dance]
```

In this case one turtle, turtle 0, performs the dance. When 0 is finished, turtle 1 performs the dance.

The list can be specific as to the order of the dance:

```
tell [0 2 1 3]
each  [dance]
```

The above commands cause the turtles to take turns but in a different order, according to the list of turtles used with tell. (**tell all** followed by **each** and commands in brackets will always result in turtle 0 performing first, followed by the other turtles in order.) **tell** and a list of turtle numbers can thus be an important controlling factor for an interesting dance.

Solo Turns

What about "solos"? In a real ballet or other dance form, an occasional soloist enters and performs alone. With LogoWriter **ask**, and a turtle number or a list of turtles, if desired (a *pas de deux* or *pas de trois*), will accomplish the same thing. What **ask** does is temporarily suspend the motion of the group long enough for the turtle addressed to perform whatever action is programmed. Once this is completed, the "dance" can resume. I like to think of **ask** as being a particularly polite interruption.

Some general rules of thumb for using these terms should be made clear at this point:

tell requires an input. It can be the number of a single turtle, a list of turtle numbers (a list in Logo means that it is enclosed in brackets), or **tell** also accepts the word **all**, which stands for all four turtles in the order 0, 1, 2, and 3.

each will refer to the last designated turtle or turtles addressed by **tell**. **each** requires a list to run, so **each**'s task must be enclosed in brackets.

ask, the polite interruption, requires a pair of inputs. **ask** needs to know the turtle or turtles who get this special treatment, either a single number or a list. The second input to **ask** is always a list to do.

Slipping and Sliding

As I introduced these ideas, exploring the potential for turtle motion with a given vocabulary of "steps" and a few words which altered the action from group or four turtle activity to single turtles, some new ideas emerged.

One of the boys asked if it was possible to move the turtle sideways without turning. This provided a wonderful chance to give a short lesson on **xcor** and, later, **ycor** as a response to his follow-up question. I showed them how to write some "slides":

```
to slide.r
setx xcor + 5     <—Move in the x-direction 5
end                  turtle steps to the right
```

```
to slide.l
setx xcor - 5     <—Move in the x-direction 5
end                  turtle steps to the left
```

Changing the x-coordinate moves the turtle the specified steps (in the example above the steps are 5, but they could be any number or a variable) in the x-direction (horizontal) only, leaving the heading unchanged. Similar moves can be programmed for vertical movement by using **ycor** and **sety**. A variable input instead of a fixed + 5 or - 5 could be used to allow more variety of motion to be programmed.

The "slides" gave rise to some wonderful inventions as well as some further group exploration. The question arose: "I want the turtle to turn AND keep moving in one direction. How can I do that?" Together we worked it out. Extend the sideways move by adding a slight move:

```
slide.r
right 10
```

If this command is repeated 36 times (remember the "total turtle trip?") the turtle will turn completely around while moving sideways. Moving one turtle in this fashion while another moves in a single straight trajectory involves students in direct confrontation with the issues of parallelism in our real-world activity and the serial nature of computer processing. We were thus required to think and talk about breaking movements down into their smallest elements in order to give a fairly successful simulation of parallel activity in the serial computer environment. The smallest procedure to move two turtles the same distance and direction, with only one turtle turning, would look like this:

```
to slide.turn
tell 0
slide.r
right 10
tell 1
slide.r
end
```

Each turtle moves the same distance, but only turtle 0 turns. Adding a input **:times** to **slide.turn** made the procedure more useful:

```
to slide.turn :times
repeat :times [tell 0 slide.r right
   10 tell 1 slide.r]
end
```

Logo Ideas -- continued

A Disappearing Act

We discovered that applying a slide to a list of turtles caused all the turtles to make a mysterious "disappearance" rather than a clever move – a chance to learn a bit more about how Logo works! When working with a list of turtles, a procedure such as **slide**, which refers to coordinates, takes a "reading" on the coordinate of the first turtle in the list. It then applies this to the entire list! All the turtles jump on top of one another, and because a turtle on top of another in effect cancels both out (think about moving a shape on top of a stamped shape: the effect is the same), the turtles all disappear. The way around this is to be sure to use the slides only with single turtles or with the command **each**. One boy took advantage of this problem to create a dance of constantly disappearing turtles.

Variations on the Theme

Still more ideas emerged. The boys began to play with color changes of the turtles or the background or both. Music was added. Dancing turtles began to interact more with one another. Finally one boy began creating an animated series of dancers, creating shapes and then choreographing a similar dance, which used all the moves he had created but also the changes to appropriate shapes.

We used this activity as a lead in to more complex programming of motion games. It has potential to lead in other directions as well. One set of turtle dancers moving in very long steps looked remarkably like a set of bugs jumping about on the surface of a pond. How about an ecological connection to the dances, adding in some randomness to the leaps about the "pond"? Crowd study, in a small way, might also be represented here. I think of William Whyte's studies of people movements in city spaces and their relationships to one another. Turtles could be programmed to move towards, and then dodge, one another. Many other turtle environments and scenarios could be created as a follow-up to the simple dances. If your students want sound to go with their dances, they might try using a tape and creating movement and color changes to synchronize with the sounds.

On with the dance!

References:

Yoder, Sharon. (1988). *Introduction to Logo Using LogoWriter*. Eugene, Oregon: ISTE. Pages 88 - 93.

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

Beginner's Corner

Star Light, Star Bright, Lots of Logo Stars Tonight!

by Dorothy Fitch

February is a great time to stargaze. The sky is clear, the air is crisp, and you can look for constellations that aren't visible at other times of the year.

This month, we'll explore many different types of stars and plan a science and math project for older students.

A Classroom Idea

Why not challenge your students to draw a star in Logo? Let them work in groups, give them paper and pencils to doodle and plan with, and, most important, make sure that they understand that there is no "right way" to draw a star. See what they come up with, and let them share their designs with the rest of the class.

Some Star Designs

Here are some star designs that you may want to explore with your students after they have had a chance to experiment on their own. They may have already come up with some of these ideas, or better ones! These are just some that are intriguing to me.

Starbursts:

This is probably the easiest type of star to draw. All you need is a REPEAT statement and a quick introduction to an important idea— the Total Turtle Trip Theorem, which Seymour Papert states in his book *Mindstorms: Children, Computers and Powerful Ideas* (published in 1980 by Basic Books, Inc. of New York).

> If a Turtle takes a trip around the boundary of any area and ends up in the state in which it started, then the sum of all turns will be 360 degrees.

"State" means that the turtle is in the same exact place on the screen and pointing in the same direction. For example, you can draw a hexagon by typing `REPEAT 6 [FORWARD 40 RIGHT 60]`. The total of 360° divided by 6 turns means that each turn must be 60°. To make a starburst, we can just add one more instruction to make the turtle back up to the center point before turning. Try these instructions:

```
REPEAT 6 [FORWARD 50 BACK 50 RIGHT 60]
```

or

```
REPEAT 8 [FORWARD 50 BACK 50 RIGHT 45]
```

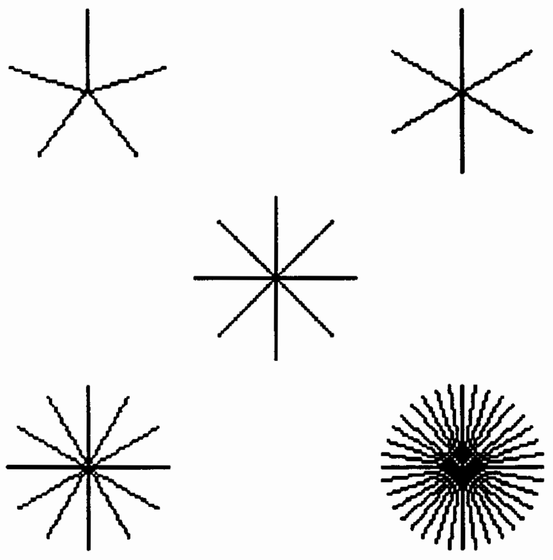
or

```
REPEAT 7 [FORWARD 50 BACK 50
        RIGHT 360/7]
```

In the last example, why work out the long division if the computer can do it for you? Better yet, design a procedure that takes as an input the number of rays you want and let the computer figure out the angle, like this:

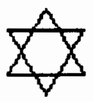
```
TO STARBURST :NUMBER
REPEAT :NUMBER [FORWARD 50 BACK 50
   RIGHT 360/:NUMBER]
END
```

Try typing STARBURST 8 or STARBURST 12 or STARBURST 50!



Star of David:

You can spend quite a while using turtle graphics commands to create a symmetrical Star of David like this:



If you are a real mathematician, you can probably devise a formula and get it right the first time. But if you are like me, you'll do it using trial and error, and there's nothing wrong with that. Here are two procedures to draw my version: STAR.OF.DAVID and TRIANGLE.

Beginner's Corner – continued

```
TO STAR.OF.DAVID
RIGHT 90
TRIANGLE
LEFT 90
PENUP
FORWARD 17
RIGHT 90
FORWARD 30
RIGHT 180
PENDOWN
TRIANGLE
END

TO TRIANGLE
REPEAT 3 [FORWARD 30 LEFT 120]
END
```

After taking a closer look at the star, however, I decided to try a different approach. For this version, I thought of the design as a hexagon with a triangle attached to each side. Here is the TRI procedure I created to draw a small triangle:

```
TO TRI
REPEAT 3 [FORWARD 10 LEFT 120]
END
```

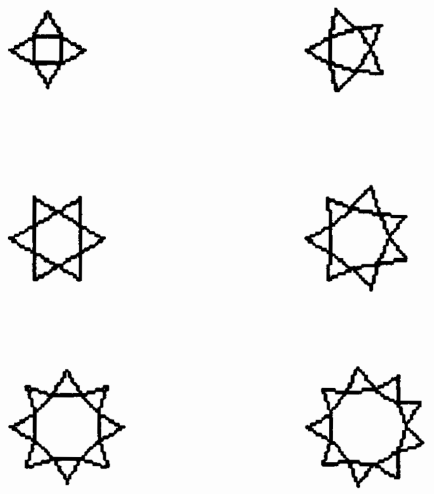
Now I can just type:

```
REPEAT 6 [TRI FORWARD 10 RIGHT 60]
```

It makes the right design, though a little tipped. That is rectified by first typing `RIGHT 30`.

Again, why not write a procedure that takes as input the number of sides we want?

```
TO POINTS :NUMBER
REPEAT :NUMBER [TRI FORWARD 10 RIGHT
   360/:NUMBER]
END
```



These six designs were created using POINTS 4, POINTS 5, POINTS 6, POINTS 7, POINTS 8 and POINTS 9.

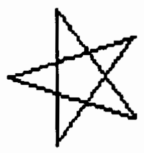
<u>Surprising Stars</u>

So if a Total Turtle Trip is 360°, can you make a five-pointed star by typing this?

```
REPEAT 5 [FORWARD 60 RIGHT 360/5]
```

No, that draws a pentagon. So, how do you make a "real" star?

To answer that question, try walking around a five-pointed star. No, you're not too old to play turtle, and neither are your students. It's not as easy as it looks to walk around a star. In fact, it's quite a humbling experience. I recommend that you spend some time practicing in the privacy of your own home before you attempt it in front of your class.



What you will discover (and what your students can discover) is that you see the four walls of the room twice before you complete the star, not just once as you would if you walked around a square or a pentagon. This means that you have turned a total of 360° times 2, or 720° before completing the shape. So, you can make a five-pointed star by typing:

```
REPEAT 5 [FORWARD 100 RIGHT 720/5]
```

or

```
REPEAT 5 [FORWARD 100 RIGHT 144]
```

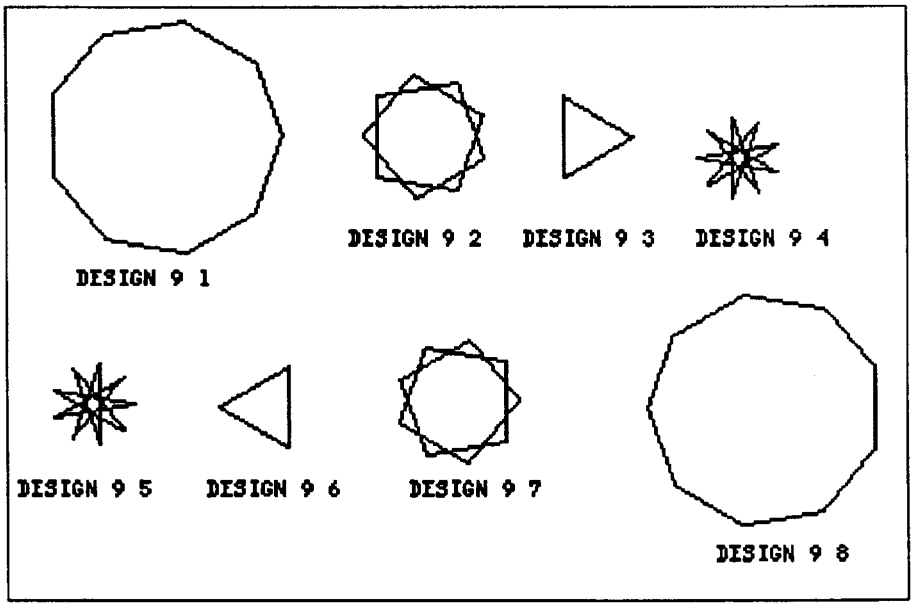
This leads to the question, what if we try other numbers of sides and other multiples of 360°?

Here is a procedure that you can use to explore this idea:

```
TO DESIGN :NUMBER :TIMES
REPEAT :NUMBER [FORWARD 50 RIGHT
   (360*:TIMES)/:NUMBER]
END
```

For example, you can type DESIGN 5 2 (5 sides times 2 Total Turtle Trips) to get the familiar five-pointed star. All right, it's a little tipped. Type RIGHT 18 before you draw it to straighten it out. If you are asking where the number 18 comes from, read on. Otherwise, skip to the next paragraph! If you type LEFT 144 after you draw the star, the turtle will point in the direction it was going before it made the last 144°

turn. Since 180 (half of the way around a circle) minus 144 is 36, it means that the angle between the lines where the turtle finishes is 36°. If you turn the turtle half of this angle (36/2 or 18°) before drawing this star, it will be pointed properly. You may just want to let your students experiment to find this magic number and them let them explain it to you!



What is interesting about this procedure is that it can give some unexpected results. For example, these designs were all created with 9 as the first input. Are any exactly alike? Why are some shaped like triangles? Why are some patterns repeated? Why are some designs facing the opposite way? Why don't they all have lots of points? Can you predict which numbers will give you regular shapes and which will give you "pointy" stars? I'll leave it to you and your students to come up with hypotheses!

A Constellation Project

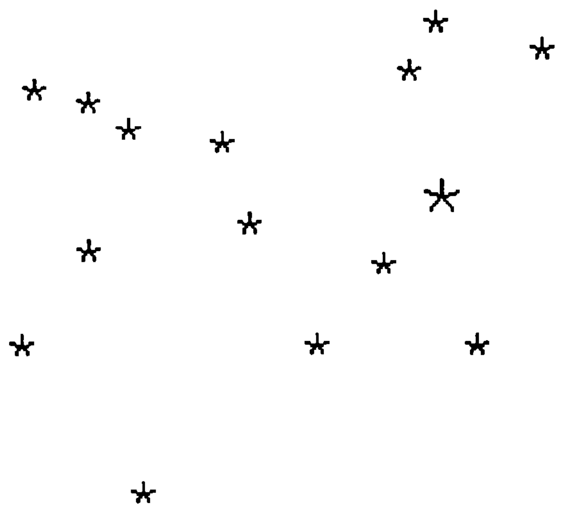
Combining stars into constellations makes an interesting project requiring both scientific research and practice in mathematics. In Terrapin's *Logo Innovations* package, Martha Carr of Gardner, Colorado suggests that students research their sign of the Zodiac and draw stars on the screen to create the constellation. Her students research the distances that separate the stars and draw appropriately sized stars to convey the proper magnitudes.

In my recent New Hampshire Audubon newsletter there is information about some winter constellations in a young reader's section. (You never know where you might find useful tidbits of information for your Logo explorations.) Provided with dots for stars, children are asked if they can connect the stars and discover animal shapes in the constellations. And, for fun, try drawing a group of stars, giving it a name and making up a story about it to share.

The newsletter describes the constellation Canis Major (or "Big Dog") as being found southeast of Orion and including Sirius, the brightest star in the sky. It is easiest to see between January and March (at least in this part of the world).

I created a couple of versions of this constellation using SETXY to place the stars on the screen (again by trial and error) until they looked just right.

Here is the pattern made by just the stars:



I found that debugging became easier when I numbered the stars, so here is my numbered version and the program that draws the stars. Note that star number 4 is Sirius, and is drawn slightly larger.

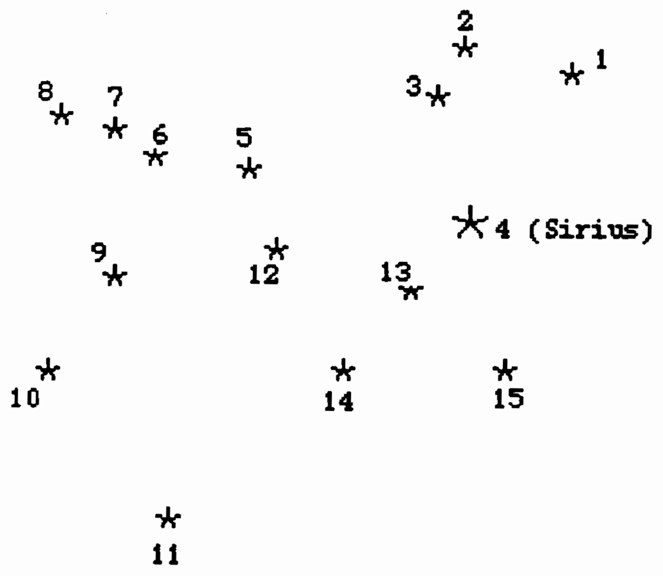
Beginner's Corner – continued

```
TO CANIS.MAJOR
PENUP
SETXY 110 75      STAR      ; 1   <—these
SETXY 70 85       STAR      ; 2    are the star
SETXY 60 67       STAR      ; 3    numbers
SETXY 72 20       SIRIUS    ; 4
SETXY -10 40      STAR      ; 5
SETXY -45 45      STAR      ; 6
SETXY -60 55      STAR      ; 7
SETXY -80 60      STAR      ; 8
SETXY -60 0       STAR      ; 9
SETXY -85 (-35)   STAR      ; 10  <-- *
SETXY -40 (-90)   STAR      ; 11
SETXY 0 10        STAR      ; 12
SETXY 50 (-5)     STAR      ; 13
SETXY 25 (-35)    STAR      ; 14
SETXY 85 (-35)    STAR      ; 15
END
```

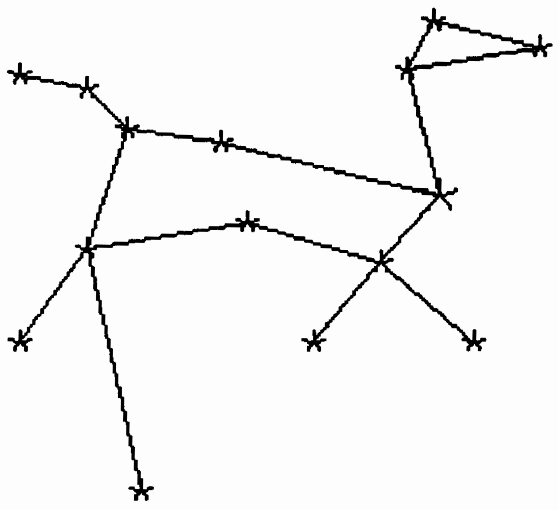
* negative second input must be placed in parentheses, or Logo will try to subtract it!

```
TO STAR
PENDOWN
REPEAT 5 [FORWARD 4 BACK 4 RIGHT 72]
PENUP
END
```

```
TO SIRIUS
PENDOWN
REPEAT 5 [FORWARD 6 BACK 6 RIGHT 72]
PENUP
END
```



Here is a version of the constellation with lines connecting the stars, as constellations are often represented in books. Challenge your students to modify the CANIS.MAJOR procedure to connect the stars with lines. They will have to remove the PENUP commands from the STAR and the SIRIUS procedures.



Let your students choose a constellation and write their own Logo program to place the stars. And don't forget to display the printouts on the bulletin board!

Happy stargazing, Logowise or otherwise!

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

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

About the Cover

The cover drawing, "Walrus," was made by Anthony Reitz, an 8th grade student of Carole Rivera. Carole runs a computer club at Everett Elementary School in Chicago, IL. Both she and her students greatly enjoy their weekly work with Logo.

Logo LinX

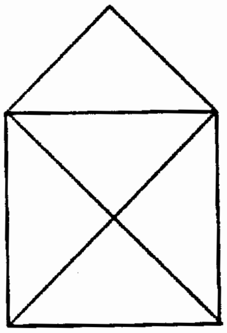
Euler and the Turtle
by Judi Harris

Let's face it, facilitators. Most of us have a Peter Pan or Wendy hiding behind our wise, adult facades, who reign in full glory whenever we sniff Logo in the classroom air. If "growing up" means losing the joyful, experimentative, problem-solving flexibility that is innately child like, then we must reserve a special corner of our personalities that will never "mature." Chances are that you have already done that, and your fondness for Logo stems partly from the opportunity it affords to exercise the "never-never land option."

Modes of Mischief

Soon spring will be in the air, and in anticipation, memories flood back of how that special vernal aroma would incite even the most studious of my childhood friends to mischief. We perfected the technique (we thought) of appearing to listen in class, while really concentrating on something quite different.

I remember specifically one puzzle "fad" that mesmerized us for several weeks. The challenge was to draw this figure without crossing a line already drawn, or lifting the pencil from the paper. How would you do that, given those constraints?

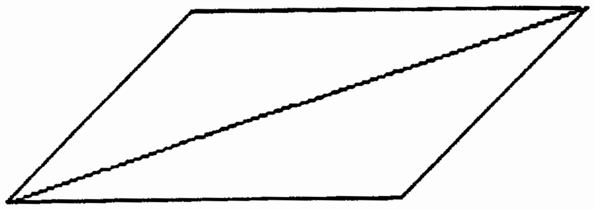


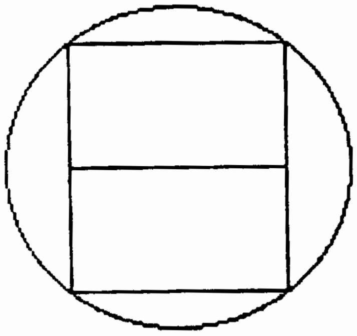
We finally did solve the problem, but the "magic" of it remained; some pictures could be drawn this way, and others couldn't. I doubt that any of us recognized the inherent instructional value of such an activity. We also would have been surprised to discover that one of the world's most prolific mathematicians pondered a similar puzzle in the early 18th century. He also identified the pattern behind the "magic."

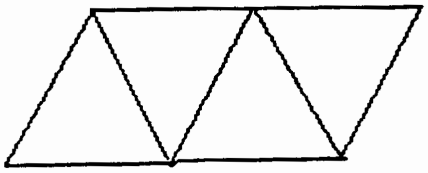
Euler and the Opus

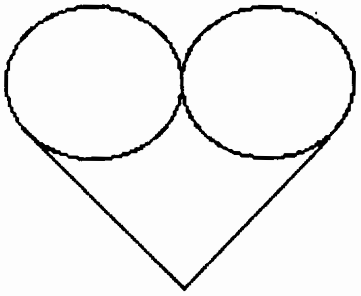
The Swiss mathematician Leonhard Euler (pronounced "oiler") had a Peter Pan in his personality long before Barrie conceived the character. Euler reportedly had 13 grandchildren and is said to have created mathematical theorems with a baby on his lap and children playing at his feet. He wrote about 800 pages per year of good quality mathematical manuscripts, and is credited with concocting the original ideas of topology.

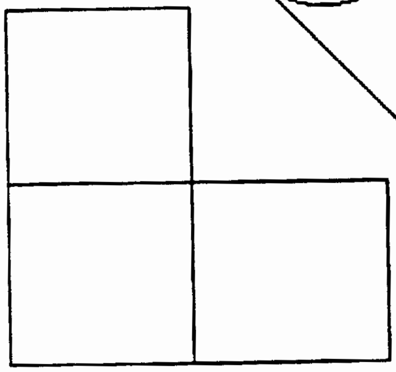
Enter the 20th century Logo turtle, and 18th century Euler line drawings like the one that intrigued us in elementary school and get a burst of Logo power. Here are some other designs which can be drawn without lifting the turtle's pen, crossing or retracing a line.





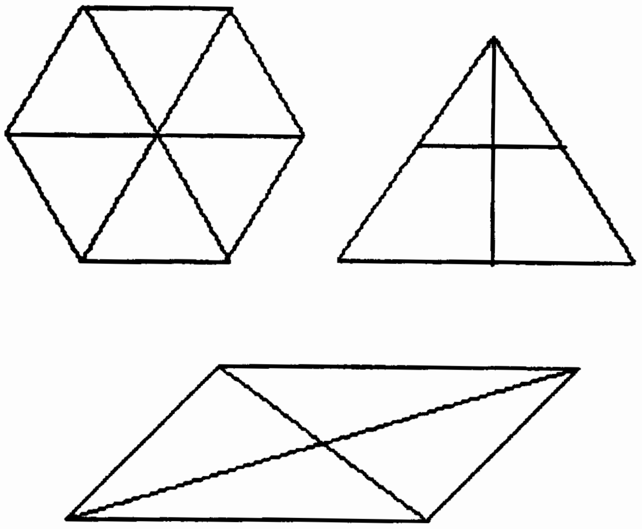






Logo LinX – continued

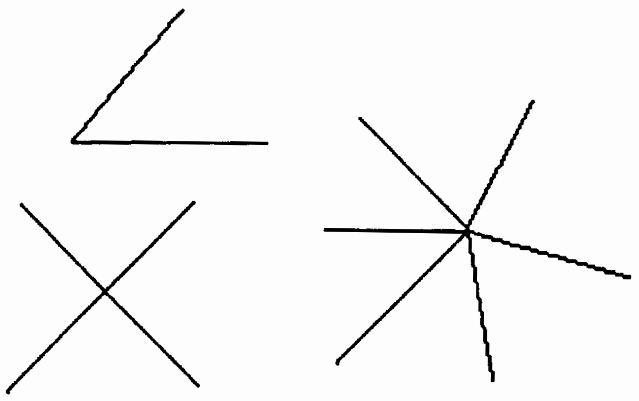
Designs such as these cannot be produced according to those procedural specifications.



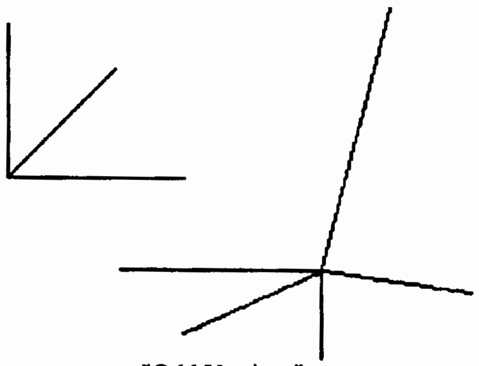
So what's the "magic"? What attributes are similar within these two groups of figures that are also dissimilar between groups? Logo students could pose, consider, discuss, revise, prove, and disprove theories after sufficient experimentation with "turtled" Euler lines. Why don't YOU stop reading for now, and call out your Peter or Wendy to help you to play with this?

Euler's Observations

Euler examined line segments extending from the vertices of polygons, and classified the intersections as "even vertices" or "odd vertices."



"Even Vertices"



"Odd Vertices"

He observed that the number of odd vertices in any figure is always even, and then went on to say:

•If there are no odd vertices in a polygon, then it can be drawn using any vertex as the starting point.

•If there are two odd vertices in a polygon, it can be drawn by starting at one odd vertex and finishing at the other without lifting the pen, crossing, or retracing a line.

•Otherwise, a polygon cannot be drawn without crossing or retracing lines, or lifting the pen.

What powerful ideas! Do you want to play again before you read on? Or did you already solve these Euler Logo puzzles?

See the Magic

Wise teachers capitalize upon children's natural interests and preferred activities. Why not make a set of polygons that can be drawn according to Euler's specifications? Challenge your students not only to solve the puzzles with the turtle, but to try to see the pattern to the "magic." Most importantly, encourage them to create Euler puzzles of their own for their classmates to solve.

Oh – by the way – if your Peter or Wendy wants to check how mine solved the Euler puzzles mentioned earlier in this article, help them to send a stamped, self-addressed envelope to me at the address at the end of the article or send an electronic request on either BitNet (JudiH@Virginia.bitnet), or CompuServe (75116,1207).

An earlier version of this article originally appeared in the March1986 issue of *The National Logo Exchange* (Volume 4, Number 7, pages 1-2)

Judi Harris
621F Madison Avenue
Charlottesville, VA 22903
CIS: 75116,1207
BitNet: JudiH@Virginia

The 1989 International Computer Problem Solving Contest

Elementary Logo Results
by Donald T. Piele

The International Computer Problem Solving Contest (ICPSC) is an annual contest for precollege students conducted at local contest sites throughout the world. Held on the last Saturday in April, it challenges teams (from one to three members each) to solve five problems in two hours using any programming language. Last spring, approximately 400 teams participated in the Elementary Logo Division (Grades 4-6). This month we present the results of the Elementary Logo Division along with the problems and sample solutions. The Senior and Elementary Logo division results appeared earlier The BASIC and Pascal contest results appear in *The Computing Teacher*.

From 1981 to 1988, the contest was sponsored by the University of Wisconsin-Parkside. Currently it is sponsored by ICPSC – a non-profit organization with financial support from USENIX, the technical association of UNIX users.

Introduction

For the first time a winning team came from a family that provides schooling for its children at home. Noah and Adam Gintis, representing the Boca Homeschoolers in Boca Raton Florida, captured first place in the Elementary Logo Division.

> Noah, age 11, has been using computers at home for as long as his mother can remember. "At the age of two, he demonstrated that he could be trusted with floppy disks, and, from that time on, he was allowed to use our equipment whenever he wished. At five, he began programming in Logo, and, by the time he could read, he taught himself Basic by taking manuals to bed with him each night. He now does all his programming in QuickBasic."

Adam, age 7, has just begun programming in Logo. "He understands the concept of structured programming, and has learned the various graphics commands", his mother wrote. In the contest, Adam worked on the graphics problems while Noah tackled the word and sentence manipulation problems – something that Adam wants to learn next.

Using the computer has become an integral part of both boys' education. Noah uses the Q&A database program to keep a list of the books he has read, when he read them, and where they are currently stored. Adam used Lotus 1-2-3 to print out addition and multiplication tables that he needed to learn. And both boys enjoy playing adventure games and are now interested in learning how to write their own. Noah spends his free time on the computer or reading science fiction. Adam, on the other hand, prefers to spend his free time with his chemistry set."

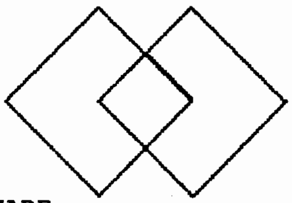
The Boca Homeschoolers is a support group for families who teach their children at home. "It provides the group experiences for the kids," Mrs. Gintis said. "Teams have been formed for various competitions such as Odyssey of the Mind and Math Olympiad in which Noah and his team members have received wards. Other group activities include the President's Physical Fitness Challenge, Language Arts Day when the students get together to read original compositions, Famous American Day when they report on an American of their choice, and Science Fair Day when they present any projects they have been working on at home."

The team of Noah and Adam Gintis was one of sixteen teams to solve all five problems in the ICPSC within the two hour time limit (out of the approximately 400 who tried). The average number of correct solutions was two.

1989 Elementary Logo Division Problems, Solutions

1. TWO SQUARES

Write a program that will produce the following design.



Solution:

```
TO TWO.SQUARES
SET.UP
SQUARE
MOVE.TO.SECOND.SQUARE
SQUARE
END

TO SET.UP
CG
HT
RIGHT 45
END

TO SQUARE
REPEAT 4 [FORWARD 50 RIGHT 90]
END

TO MOVE.TO.SECOND.SQUARE
FORWARD 25
LEFT 90
BACK 25
LEFT 90
END
```

ICPSC Contest -- continued

2. DOWN

Write a program called DOWN that removes letters one at a time from the end of any word. For example, if the word were LOGO, and you typed

```
DOWN "LOGO
```

the program would print

```
LOGO
LOG
LO
L
```

Test your program by typing

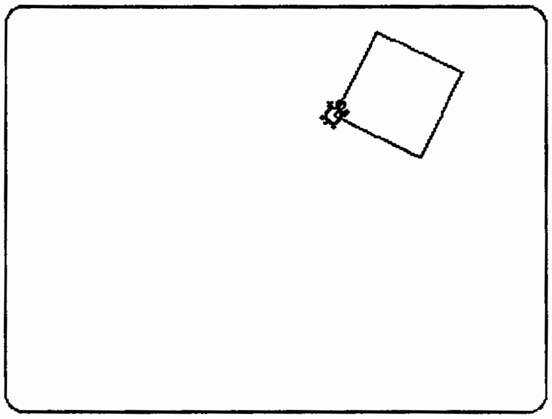
```
DOWN "PROGRAM
```

Solution

```
TO DOWN :WORD
IF :WORD = " [STOP]
PRINT :WORD
DOWN BUTLAST :WORD
END
```

3. RANDOM SQUARE

Write a program that will place a square at a random spot on the screen. Each time you run your program, the square should appear (as below) at a different spot on the screen. It is OK if your square wraps around the top or the bottom of the screen. Run your program four times to demonstrate that it works.



Solution

```
TO RANDOM.SQUARE
CG
ST
PENUP
RIGHT RANDOM 360
FORWARD RANDOM 100
PENDOWN

SQUARE
END

TO SQUARE
REPEAT 4 [FORWARD 50 RIGHT 90]
END
```

4. COUNTEM

Write a program that will accept a list of numbers and count the negative numbers in the list. For example, if you type

```
COUNT.EM [1 -3 -5 -3 4 7 0 -2 -9]
```

the program should print

```
5 NEGATIVE NUMBERS
```

Test your program using the list given above.

Solution

```
TO COUNT.EM :LIST
COUNT.NEGS :LIST 0
END

TO COUNT.NEGS :THE.LIST :TOTAL
IF EMPTYP :THE.LIST [(PRINT :TOTAL
   [NEGATIVE NUMBERS]) STOP]
IF ( FIRST :THE.LIST ) < 0 [MAKE
   "TOTAL :TOTAL + 1]
COUNT.NEGS BUTFIRST :THE.LIST :TOTAL
END
```

5. COIN FLIP

Write a program to simulate flipping a coin five times. Your program should print "HEADS" each time a heads is picked and "TAILS" each time a tails is picked. That is, your output might look like this:

```
HEADS
TAILS
TAILS
HEADS
TAILS
```

Run your program three times to demonstrate that it works.

Solution

```
    TO COIN.FLIP
    REPEAT 5 [FLIP]
    END

    TO FLIP
    IFELSE ( RANDOM 2 ) = 0 [PRINT
        [HEADS]] [PRINT [TAILS]]
    END
```

Something for Everyone

The major emphasis of the ICPSC contest is on the local contest site. Each year we receive many letters from local contest directors describing successful contests – some with awards ceremonies and banquets – where the winning team may have solved three problems. If we do our job right, everyone can solve some of the problems, and the very best in the world will be challenged to solve all five. The ICPSC provides the problems, sample solutions, instructions for the director and local judges–everything to make it easy for local teachers and administrators to conduct a successful contest. Our goal is simple – to promote the development of computer problem solving skills. Come join us!

1990 Contest

The 10th Annual ICPSC will be held on Saturday, April 28, 1990, with Friday April 27, and Monday, April 30 as alternate contest dates. For more information on how your school or school district can become a contest site, send your request to the address below. You will also receive a free copy of *Compute It!*, the newsletter of the ICPSC.

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

1989 Elementary Logo Division Rankings

Rank	Team	School	City, State/Country	Director	Advisor
First	Noah Gintis Adam Gintis	Boca Homeschoolers	Boca Raton, Florida	Barbara Gintis	Barbara Gintis
Second	Sam Thompson Jason Hornik	Kendale Lakes	Miami, Florida	Sharon Freedman	Sharon Freedman
Third	David Stevenson David Murray Robert Mazuch	St. Michael's University Middle School	Victoria, B.C., Canada	Dr. A.D. McMaster	
Fourth	Daniel Wolfe Eddie Lee Nils Barth	Roeper City and Country School	Bloomfield Hills, Michigan	Terry Rudman	Terry Rudman
Fifth	Chad Trost Tom Kennedy Geoffrey Benson	Galtier Magnet School	St. Paul, Minnesota	Mike Amidon	Paul Krocheski
Sixth	Gavin Hurley	Scoil An Spioraid Naoimh	Bishopstown, Cork, Ireland	Michael D. Moynihan	
Seventh	Brook Porter Matthew Ball Eric Hartmeister	Mitchell School	Golden, Colorado	Forrest E. Smith	Joe Sewall
Eighth	Stephen Fuller	Scoil An Spioraid Naoimh	Bishopstown, Cork, Ireland	Michael D. Moynihan	
Ninth	John-Paul Corkery	St. Joseph's N.S.	Bishopstown, Cork, Ireland	Michael D. Moynihan	
Tenth	Eoin Curran	Bishop Galvin School	Dublin, Ireland	Dr. John S. Close	Dr. John S. Close
Eleventh	Travis Kopp	Governors Ranch School	Littleton, Colorado	Forrest Smith	
Twelth	Paul Sturm Peter Lunseth Kevin Behrens	Oxbox Creek Elementary	Champlin, Minnesota	Mike Amidon	Diane Hewitt
Thirteenth	Matthew Scully	St. Aidan's C.B.S.	Dublin, Ireland	Dr. John S. Close	Dr. John S. Close
Fourteenth	Timothy Deegan	Lindsay Rd School	Dublin, Ireland	Dr. John S. Close	Dr. John S. Close

MathWorlds

**edited by
A. J. (Sandy) Dawson**

I am pleased to welcome Jim King back as a columnist. A couple of years back Jim wrote the highly provocative and stimulating article about the Affine Turtle. Once again Jim has come up with ways of turning routine turtle geometry ideas into inventive means of exploring new visualization challenges involving the use of negative numbers in Logo procedures. These challenges give rise to very interesting questions regarding symmetry of figures.

The Power of Negative Thinking

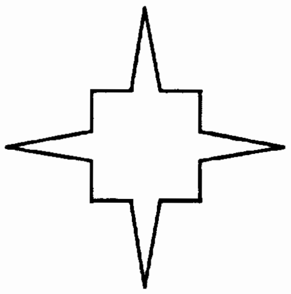
by Jim King

One of the milestones in the development of mathematics occurred when it was realized that the counting numbers 1, 2, 3, ... could be extended by considering zero and the negative numbers -1, -2, -3, ... to be full-fledged numbers as well. All the operations of arithmetic can be extended to include these new numbers in such a way that the usual rules, such as (a + b)c = ac + bc, continue to hold (the need to preserve these rules is why the product of two negative numbers has to be positive). Far from being an ivory-tower abstraction, the negative numbers allow us to express mathematically such varied scientific and practical concepts as negative electrical charge and negative balance of trade.

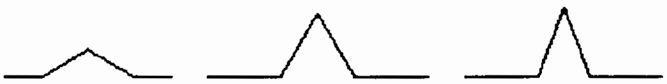
Negative numbers are important in turtle geometry, too. It may seem that since FD -50 and RT -60 are really synonyms for BK 50 and LT 60, it is not important that FD and RT (and BK and LT) can take negative inputs. After all, if you want to go backwards, you can just type BK! But this is not always true; we may design a turtle geometry procedure by thinking about what the turtle will do going FD by a positive number; but when we run it the FD command may get a negative input instead. This can lead to unexpected and interesting results. The procedure may draw a whole constellation of figures that we would scarcely have been able to imagine at the outset. The power of general ideas in turtle geometry (such as the rule of 360) is such that a procedure that draws a figure with a certain symmetry will often continue to draw such figures even though the inputs are not at all what we originally visualized.

What's a bump on a square?

Let's draw a square with a triangular bump on each side like this one.



We begin with a procedure BUMP that draws simple bumps like these:



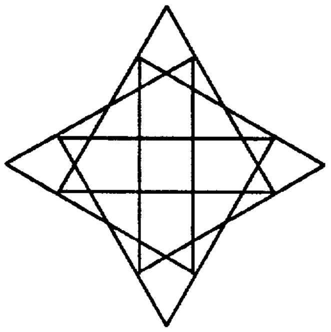
and then arrange them around a square as in the first figure using the procedure BUMPSQ.

```
TO BUMP :STEP :BSIDE :PTTURN
FORWARD :STEP
LEFT :PTTURN/2
FORWARD :BSIDE
RIGHT :PTTURN
FORWARD :BSIDE
LEFTT :PTTURN/2
FORWARD :STEP
END

TO BUMPSQ :STEP :BSIDE :PTTURN
REPEAT 4 [BUMP :STEP :BSIDE :PTTURN
   RIGHT 90]
END
```

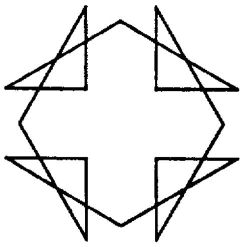
This seems a simple enough procedure. We can change the shape by varying the inputs, for example :PTTURN is the turn at the tip of the bump, and the sides of the bump are BSIDE. The shapes are rather easy to visualize, we can see that a :PTTURN of 160 gives a sharper point than one of 120 and that bigger values of :STEP lengthen the sides of the square without changing the shape of the bump.

So now that we understand BUMPSQ,.we can ask the question: is the cross-in-a-star figure below a bump on a square?

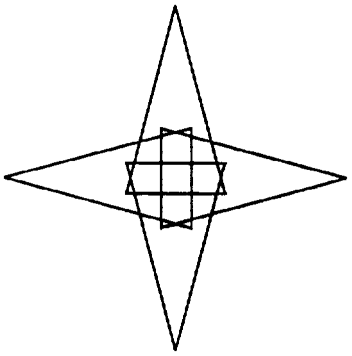


It does not look like what we described above, but in fact it is also drawn by BUMPSQ. This procedure takes a turn into the twilight zone when we use negative inputs. It takes a lot more effort to visualize the results of such inputs, and the shapes can get rather strange.

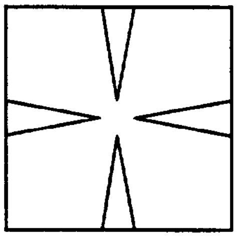
Here are some examples of BUMPSQ with :SIDE negative and the two other inputs positive; the figures were drawn by BUMPSQ -30 50 60



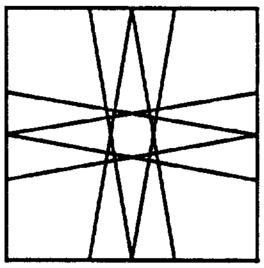
and BUMPSQ -30 90 150.



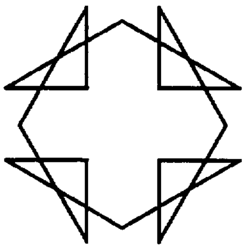
Next are some examples when :BSIDE is negative and the other inputs are positive. The first two were drawn by BUMPSQ 60 -45 160



and BUMPSQ 80 -120 160.



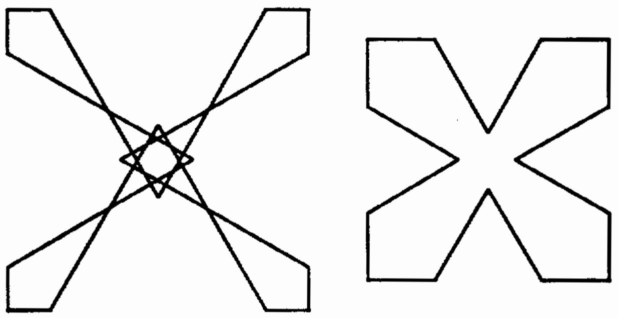
The last figure, which was drawn by BUMPSQ 30 -50 60, is congruent to the one drawn by BUMPSQ -30 50 60.



Finally, we can change the sign of :PTTURN. This has the effect of turning the right turns into lefts and the lefts into rights, so that BUMP 20 20 -120 looks like BUMP 20 20 120 except that the former is on the right side from the turtle's point of view and the latter is on the left.

MathWorlds – continued

Here are some examples drawn by BUMPSQ 25 40 -120 and BUMPSQ 16 80 -120.



This shows how changing the sign of the inputs goes far beyond the initial conception of a bumpy square.

•Explore BUMPSQ and pick out your favorite shapes. Can you duplicate the cross-in-a-star figure above?

•Write a bumpy N-gon procedure that puts bumps on regular polygons and explore the shapes that this procedure draws. Notice that the BUMPSQ procedure produces figures with 4-fold rotational symmetry and also with eight mirror symmetries. What symmetries does your bumpy N-gon procedure have?

•We have seen the results of changing the sign of :STEP or :BSIDE. What happens if we change the sign of both of these inputs? In other words, what is the relation of the figure drawn by BUMPSQ 30 90 150 to that drawn by BUMPSQ -30 -90 150? Or compare the results of BUMPSQ -30 90 150 with those of BUMPSQ 30 -90 150? Can you make a general statement that explains what is going on here? Can you explain it in general?

•What kind of figures are drawn by BUMPSQ when either :STEP or :BSIDE is zero?

Negative thinking

The BUMPSQ examples illustrate one aspect of the power of negative thinking.

Negative Thinking Slogan # 1. Always consider using negative inputs, especially where they are unexpected.

Ask yourself what a procedure will do if you change a positive input to a negative (or zero) input. Your aim can be the discovery of a pretty or visually surprising figure hidden in an old procedure. You can also think about some geometrical points: How are the symmetries of the figures with positive inputs related to those with negative inputs? Which inputs give congruent figures?

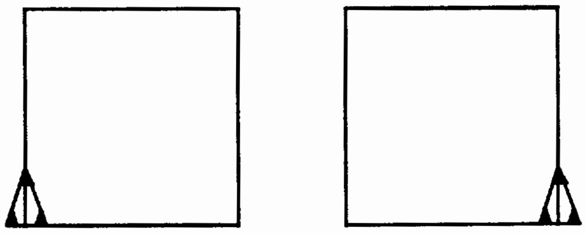
A second aspect of negative thinking in Logo is to plan for the possibility of negative inputs when you write a procedure. For example, it is very common to need two N-gon procedures, one for counterclockwise N-gons and one for clockwise N-gons:

```
TO LNGON :N :SIDE
REPEAT :N [FORWARD :SIDE LEFT 360/:N]
END

TO RNGON :N :SIDE
REPEAT :N [FORWARD :SIDE RIGHT 360/
   :N]
END
```

When :N is 4, these become right and left squares:

However, by thinking negatively, you can do both jobs



with one procedure:

```
TO LNGON :N :SIDE
REPEAT ABS :N [FORWARD :SIDE LEFT
   360/:N]
END
```

Now LNGON 5 40 draws a counterclockwise pentagon and LNGON -5 40 draws a clockwise pentagon, so RNGON is not needed.

The absolute value function ABS outputs the absolute value of a number, i.e., ABS 4 is 4 and ABS -5 is 5. If your Logo does not have ABS built-in, you can add it; here are definitions in LCSI and Terrapin syntax.

```
TO ABS :NUM
IF :NUM < 0 [OUTPUT (-:NUM)]
OUTPUT :NUM
END
```

```
TO ABS :NUM
IF :NUM < 0 THEN OUTPUT (-:NUM)
OUTPUT :NUM
END
```

Negative Thinking Slogan # 2. Always try to write procedures so they will accept negative inputs.

Thinking negatively applies to star polygons as well as N-gons. This procedure will work with negative :P and/or :Q.

```
TO LPQGON :P :Q :SIDE
REPEAT ABS :Q [FORWARD :SIDE LEFT
   :P*360/:Q]
END
```

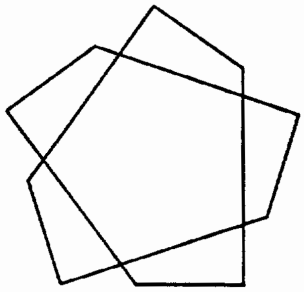
Arcs with negative angles

Now this ambidextrous N-gon is nice enough, but the place that Slogan #2 really shines is with arcs. All the versions of the Logo arc procedure that I have encountered (I may have missed some) fail to work with negative angle input. This severely hobbles the procedures for some purposes.

For example, suppose we wish to make a scissors figure with arcs. First, let's recall how a scissors works.

```
TO LPQSCISSORS :P :Q :SIDE1 :SIDE2
   :ANG1
REPEAT ABS :Q [FORWARD :SIDE1 LEFT
   :ANG1 FORWARD :SIDE2 LEFT (:P*360/
   :Q) - :ANG1]
END
```

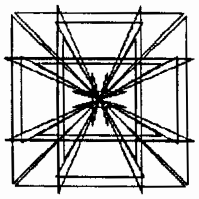
The scissor is what is inside the brackets. The figure that this procedure draws has the skeleton of a (p,q) star polygon (i.e., if you draw segments between consecutive corresponding points on the figure you get a star polygon). This works because the total turn of the scissor is :P*360/:Q. For example, here is the figure drawn by LPQSCISSORS 2 5 80 40 54. Notice that since 2*360/5 = 144, that the second LT is 144 - 54 = 90 degrees. You can easily see the 90 degree turn in the figure.



How to add spice to your lessons

```
TO ADD.SPICE
  Buy Logo Innovations.
  Pick one of 18 projects.
  Use it with your class today.
  Show others the neat things you
    can do with Logo.
END
```

Logo Innovations is a spice that can perk up your classroom lessons. While other Terrapin products focus on one subject in depth, Logo Innovations is the seasoning that will complement any curriculum.



The design at left was generated using the Mandala activity. This mandala is a random symmetrical design, a perfect Logo application.

Choose from 18 Logo Innovations activities

Logo Miniature Golf—teach estimation and strategy
Astronomy—create constellations using Logo
Logo Weather Station—connect your computer to the outside world and monitor weather conditions
Proportions—practice ratios using triangles
Little Turtle Goes to a Party—introduce young learners to directions through a delightful story
Vectors—use simple Logo commands to add vectors

Plus 12 more projects to explore!

The double-sided disk contains 19 ready-to-use programs, and the 32-page resource guide includes three off-computer activities.

See what other teachers are doing with Logo—order your copy today!

Terrapin Software (207) 878-8200
400 Riverside Street Portland, ME 04103

Name______________________________

Address ____________________________

City ____________State____ Zip______

___ I am enclosing a check to Terrapin for $14.95.

Please check the version of Logo you have:

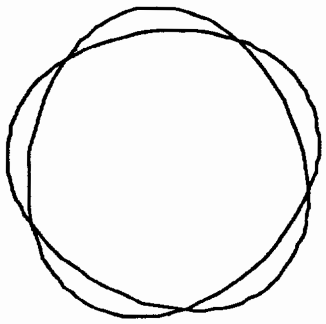
__ Terrapin Logo for the Apple __ Logo PLUS

MathWorlds – continued

Now we can use the same idea to draw similar figures with arcs. We substitute the instruction LARC :RAD :ANG for FORWARD :SIDE LEFT :ANG in the scissor. This gives this new procedure with arcs. (The funny name is part of quasi-systematic scheme that I use for naming procedures based on (p,q)-gons; the LALA is for LArc LArc.)

```
TO PQGON.LALA :P :Q :RAD1 :RAD2 :ANG1
REPEAT ABS :Q [LARC :RAD1 :ANG1 LARC
    :RAD2 (:P*360/:Q) - :ANG1]
END
```

Here is an example drawn by PQGON.LALA 2 5 80 40 54; notice that :ANG1 and :P and :Q are the same as in the previous example, so the arcs have angles 54 and 90.



Announcing

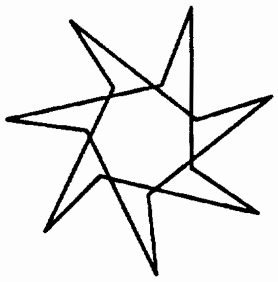
Teaching and Learning in Logo-based Environments

edited by
G. Schuyten and M. Valcke

This proceedings of the EuroLogo conference will be available during the 1st quarter of 1990. The cost is $60.00. Write to

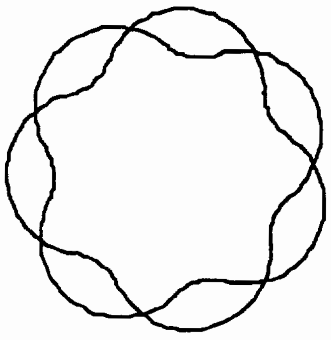
Tim Conovan
IOS
North American Operations
Postal Drawer 10558
Burke, VA 22009
USA

Now let's contrast this with another example where negative numbers appear unexpectedly. This time we draw a figure with LPQSCISSORS 2 7 60 30 160.

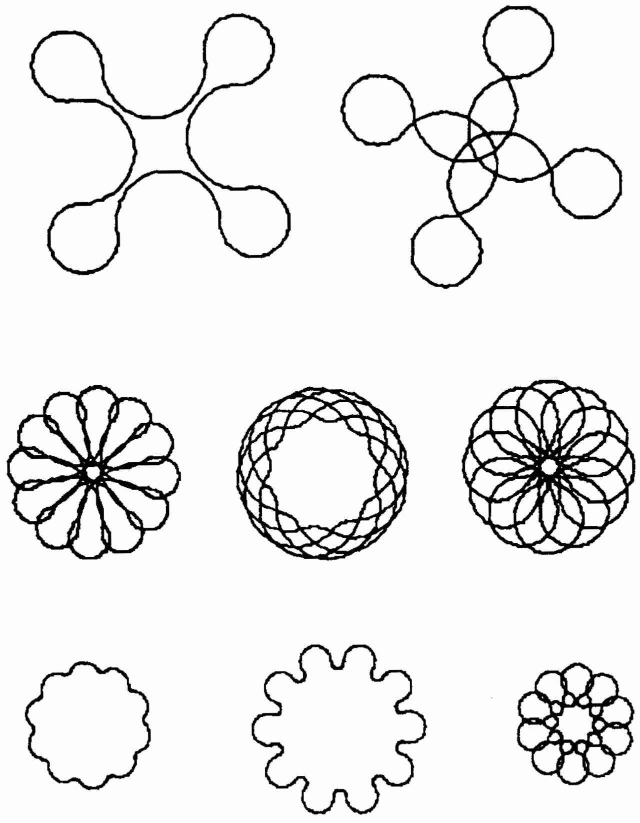


Notice this time that while there is a left turn of 160 degrees in the figure there is also a right turn, although the procedure has two LT commands and no RT. The reason is that the second turn is LT (2*360/7) - 160, which is approximately 103 - 160, a negative number. This is an important feature that makes LPQSCISSORS so versatile; the procedure automatically computes the second angle and draws the figure given the first angle, no matter whether the second angle is positive or negative.

However, if we try the same :ANG1 and :P and :Q in the PQGON.LALA procedure, we get in trouble if our LARC procedure does not work with negative angle input. One cure would be to make a more complicated form of PQGON.LALA with an IF statement which would test for the sign of the second angle and then branch to a RARC right arc procedure in this case. However, it is simpler to build this into the LARC procedure in the first place. Here is the figure drawn by PQGON.LALA 2 7 30 20 160 with such a LARC procedure; notice the right-ward turning arcs.



Here are some other examples drawn by this procedure; can you figure out the values of :P and :Q and the approximate value of the other inputs?



Defining arc procedures that accept negative angles

Once you have the idea, it is easy to make your arc procedure accept negative angles. Here are the procedures that I use.

```
TO LARC :RADIUS :ARCANGLE
IF :ARCANGLE < 0 [RARC :RADIUS
   (-:ARCANGLE)  STOP]
REPEAT INT :ARCANGLE/5 [FORWARD
   :RADIUS * 6.2918/72 LEFT 5]
FORWARD (:ARCANGLE/5 - INT
   :ARCANGLE/5) * :RADIUS * 6.2918/72
LEFT :ARCANGLE - 5 * INT :ARCANGLE/5
END
```

```
TO RARC :RADIUS :ARCANGLE
   IF :ARCANGLE < 0 [LARC :RADIUS
      (-:ARCANGLE)  STOP]
   REPEAT INT :ARCANGLE/5 [FORWARD
      :RADIUS * 6.2918/72 RIGHT 5]
   FORWARD (:ARCANGLE/5 - INT
      :ARCANGLE/5) * :RADIUS * 6.2918/72
   RIGHT :ARCANGLE - 5 * INT :ARCANGLE/5
   END
```

These are standard arc procedures except there is an additional first line beginning with IF. This simply calls the other arc procedure when the angle is negative. The last two lines of these procedures adjust for the case when :ARCANGLE is not evenly divisible by 5. Other forms of LARC and RARC make the correction differently or leave it out; that does not affect the adaptation for accepting negative angle inputs; just put the line with the IF in your favorite arc procedure (making minor modifications as necessary to allow for your brand of Logo and the name of your arc procedure). Be sure to leave no space between the - sign and :ARCANGLE or you will probably get an error.

Other areas for negative thinking

Once you have the idea, you can see negativity everywhere. For example, let the increments in POLYSPI be negative. Whenever you have a ratio in a procedure, check out what a negative ratio will do. Have fun with negative thinking, and don't forget that every silver lining has a cloud!

Note: Some of this was adapted from *Logo Geometry: A College Geometry Course* © James R. King 1986, 1987, 1988, 1989.

James King may be contacted at the Department of Mathematics GN-50, University of Washington, Seattle, WA 98195 or via email as king@math.washington.edu

A. J. (Sandy) Dawson is a member of the Faculty of Education at Simon Fraser University in Vancouver, Canada. He can be reached through Bitnet as userDaws@SFU.BITNET

Logo & Company

A Logophile's Introduction to Hypercard
by Glen L. Bull and Gina L. Bull

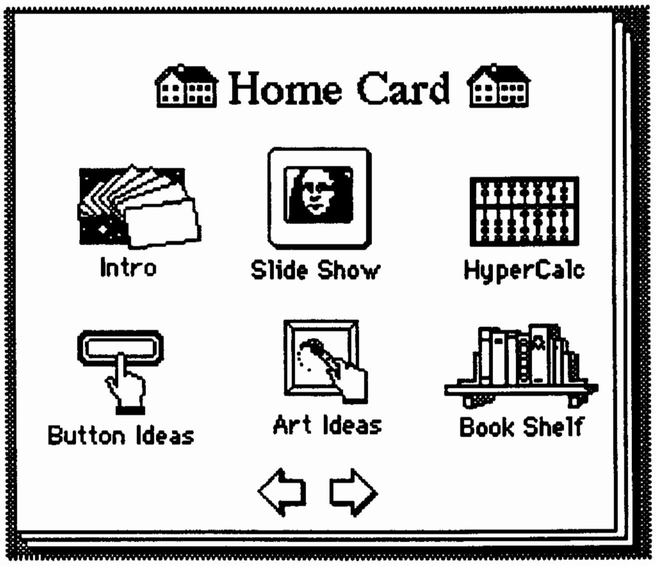
It has been a number of years since our visit to the Atari Logo laboratory near the Legal Seafoods restaurant in Cambridge, Massachusetts. However, that one visit made a lasting impression that remains today. At that time microcomputers were new and Logo was in its heyday. The Atari laboratory was stuffed full of marvelous, thought-provoking gadgets. One of the most interesting displays consisted of a Logo computer with a touch screen. There were pictures of buttons with Logo commands on the screen. If you pressed the **FD** button the turtle went forward and if you pressed the **RT** button the turtle turned right. You could also drag the buttons from place to place on the screen with your finger. We were told that these were called "sticky buttons" because you could move them from place to place on the screen. Shortly thereafter, Atari underwent a change of ownership, and none of these marvelous inventions ever found their way into commercial versions of Logo. However, years later we were surprised to find similar concepts in the Macintosh program, HyperCard.

In this column and the next we will show you how to create a "sticky button" in HyperCard similar to the ones used in the experimental version of Atari Logo. In next month's column we will show you how to create a few graphics commands to control these HyperCard objects in much the same way that the turtle is controlled with commands such as FD and RT in Logo.

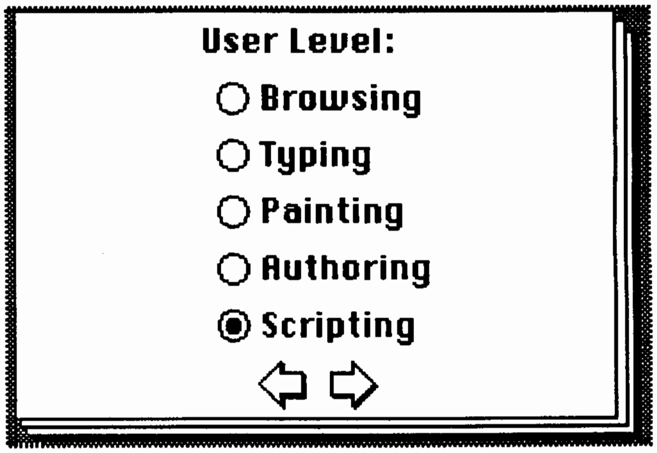
You will need access to a Macintosh computer to play with these ideas. HyperCard is provided with every Macintosh sold, just as a copy of BASIC is provided with Apple II and IBM computers. A Macintosh with one megabyte of memory is needed to run HyperCard. If you have not used a Macintosh computer before, we suggest that you run the "Guided Tour of the Macintosh" disk first. It will teach you about the conventions of using the mouse such as "dragging" and "clicking".

Once you are familiar with the Macintosh conventions you will be ready to run HyperCard. You can start HyperCard by double-clicking on the HyperCard icon:

HyperCard is based on the metaphor of a series of 3 x 5 inch index cards. When you first run HyperCard, the Home Card will appear. Clicking on any of the pictures on the Home card will run a HyperCard program. Each of these pictures, or "icons" as they are called in Macintosh terminology, will run a HyperCard program. If you would like to explore HyperCard, you can click on the *Intro* icon in the upper left-hand corner of the Home card.

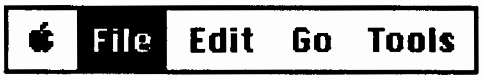


The Home card is top card on a deck of 3 x 5 inch index cards. You can move to other cards in the deck by clicking on the arrows at the bottom of the screen. Click on the *left* arrow to go to the User Preferences card, which allows you to set the *user level*. You would like to be able to access the *scripting* level, which will allow you to write HyperCard programs. If the user level is not already set to scripting, click on the button beside "Scripting" as shown below.

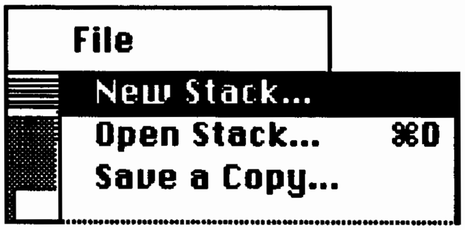


User Level:
- ○ Browsing
- ○ Typing
- ○ Painting
- ○ Authoring
- [✓ Scripting]

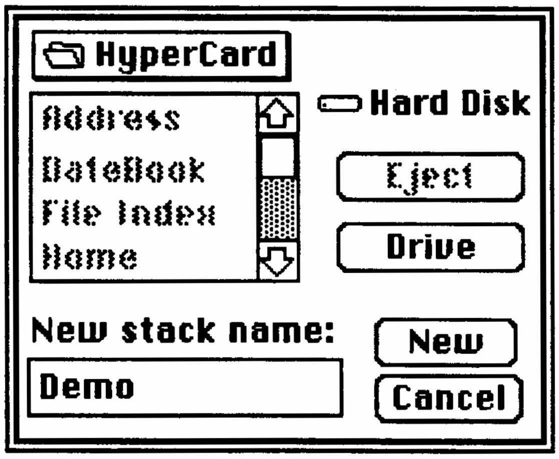
Once you have set the user level to scripting, click on the *right* arrow of the User Preferences card to go back to the Home card. At the top of the Home card you should see a Menu Bar with several options such as "File", "Edit", "Go", etc. (If the Menu Bar is not visible, you can toggle it on by holding down the **Command** key and pressing the spacebar.)



If you place the mouse pointer on the **File** option and hold the mouse button down, you should see a menu of various File Options appear. While you continue to hold the mouse button down, drag the mouse pointer down until the "New Stack" option is highlighted, as shown below.

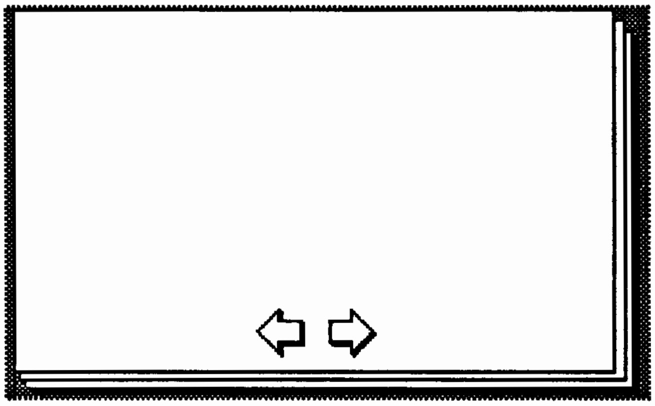


If you release the mouse button while the **New Stack** option is highlighted the following dialog box will appear. This dialog box will allow you to type in the name of a new HyperCard *stack* that you can use for experimentation. We named our stack "Demo", but you can choose any file name that you like for yours.



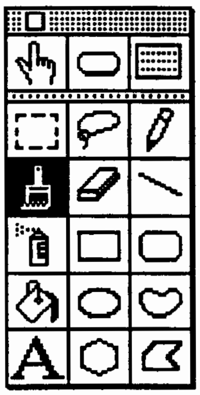
New stack name: Demo

Once you have typed in the name of your new stack, as shown above, click on the "New" button in the dialog box to create a new HyperCard Stack. This will create a new stack of electronic cards that you can use for experimentation.



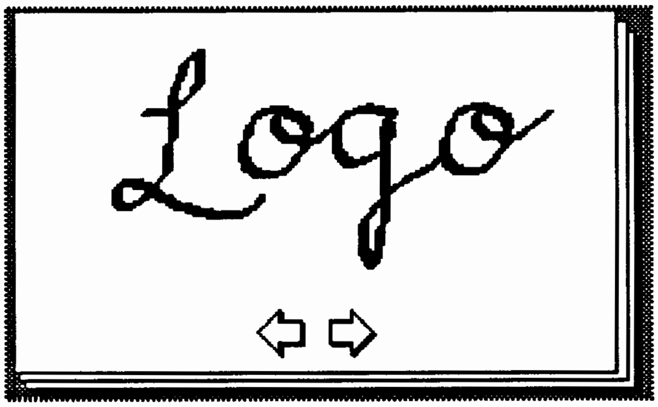
One of options on the menu bar at the top of the screen is "**Tools.**" If you place the mouse pointer on the Tools option and press the mouse button the **Tools** palette will appear. While you continue to hold down the mouse button, drag the mouse pointer to the middle of the screen. The tools palette will follow the mouse pointer. Position the **Tools** palette in a convenient location at one side of the screen.

All of the tools below the first row of the **Tools** palette are "painting" tools. For example, the highlighted tool that looks like a paintbrush is called the "**brush tool.**" The tool beside the brush is an eraser, while a spray can and a paint bucket are found just below the brush tool.

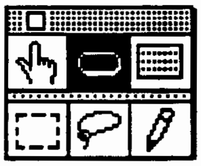


Logo & Company – continued

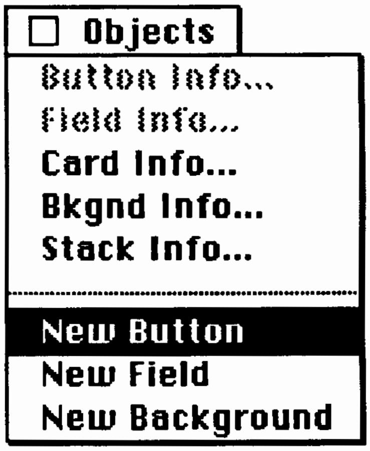
Place the mouse pointer on the brush tool and click once to select it. The brush tool will be highlighted as shown above. By moving the mouse pointer across the screen, you can draw with the brush tool just as you might with an actual paint brush. Paint flows if you hold down the mouse button as you move the brush across the screen



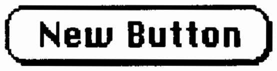
A column could be devoted to use of the graphics tools themselves. However, in this issue we will go on to show you how to create a button in HyperCard. There are three general purpose tools on the top row of the Tools palette. The icon which looks like a hand is called the "Browse tool." The object in the middle is called the "Button tool", and the object at the far right of the top row is called the "Field tool." Since we are going to work with HyperCard buttons, place the mouse pointer on the Button tool and click once to highlight it as shown below.



To create a new Button, place the mouse pointer on the Objects option on the menu bar at the top of the screen and depress the mouse button. The Objects menu should appear. As you continue to hold down the mouse button, drag the mouse pointer down the Objects menu until the New Button option is highlighted as shown below. Release the mouse button at that point, and a new button should appear on the screen.

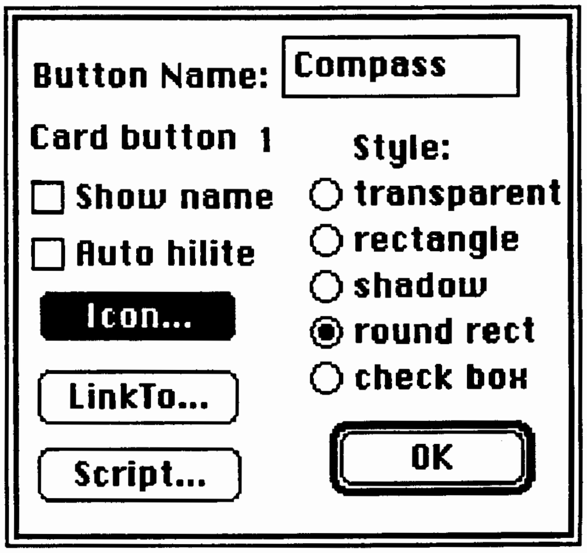


The new button on the screen will look like this. Place the mouse pointer on the button on the screen that you have just created, and double-click the mouse.

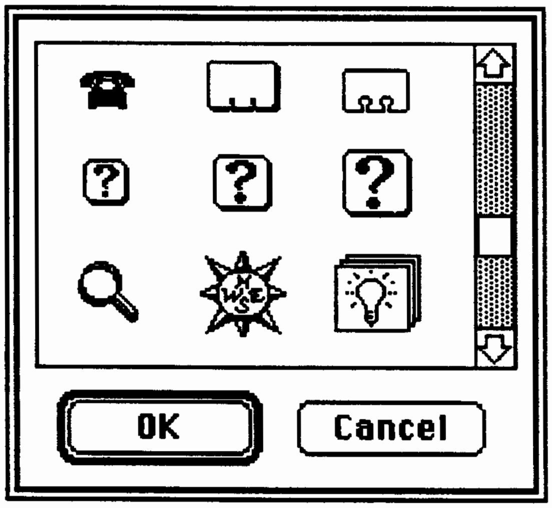


This action will place you in the Button Info dialog box, as shown below. There are three actions which you should take in this dialog box.

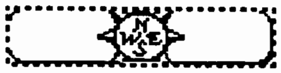
1. First, change the name of the button from "New Button" to "Compass."
2. Then click the "Show Name" check box to deselect the "Show Name" option.
3. Finally, click "Icon" button (highlighted below).



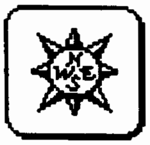
The third action (clicking the Icon button) will take you to an Icon menu similar to the one shown below. Use the elevator bar on the right-hand side of the box to scroll down through the different icons, or pictures, until the icon of the compass appears. (This icon is between the magnifying glass and light bulb icons.)



Click once on the compass icon to highlight it. Then click OK to install this icon in your new button. Initially the icon will be larger than the boundaries of the button, and will look something like this.



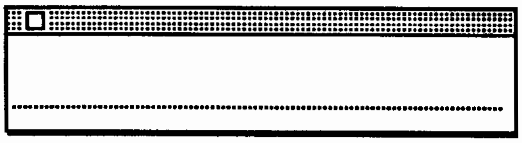
However, you can place the mouse pointer on the corner of the button, and drag the boundaries into a different shape as you hold down the mouse button. When you are finished, it should look like this.



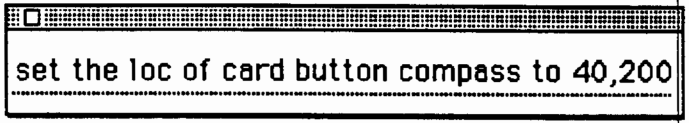
You can use the mouse to drag your compass button around the screen. Place the mouse pointer in the middle of the icon, and hold down the mouse button as you drag the compass around the screen. The compass will, in short, act much like one of the "sticky buttons" from the experimental versions of Atari Logo, except that you will be using the mouse to move the compass around rather than your finger.

You can also use programming commands to move the compass around the screen. Most versions of Logo have some form of a split graphics screen. The turtle moves around the top half of the screen, while Logo commands which control the turtle can be typed in a command center in the lower half of the screen. If you have been using Logo for any length of time, you probably don't even think about the split-screen mode, but just take it for granted.

The equivalent place for typing commands in HyperCard is called the "Message Box." You can make the Message Box appear by holding down the command key (the key to the left of the spacebar with a cloverleaf on it) and pressing "M". Typing Command-M will cause the Message Box to appear. (If you type Command-M a second time, the Message Box will disappear.)



Move the mouse pointer into the Message Box. As the mouse pointer moves into the message box, it will turn into an I-Beam shaped cursor. Click the mouse once and then type the following.

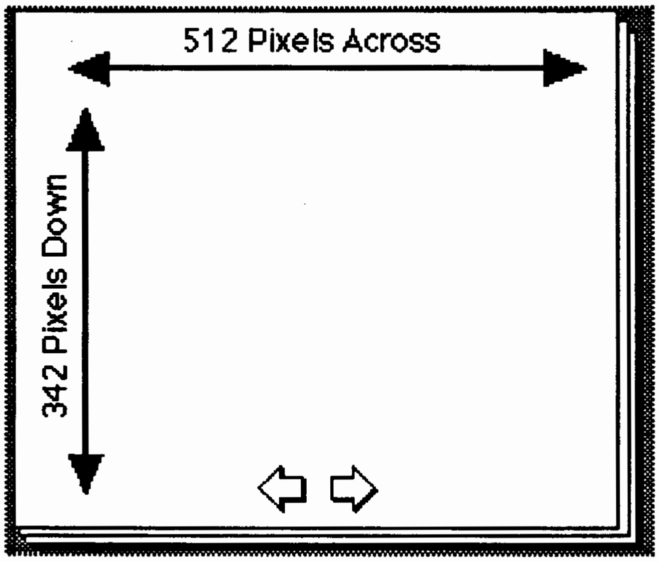


```
set the loc of card button compass to 40,200
```

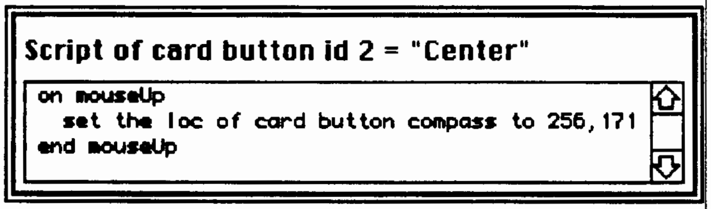
Did the compass move to the left of the screen about half-way down? This command says to do the following, "Set the location of the card button 'Compass' to a point 40 pixels over and 200 pixels down." (A pixel, or "picture element" is the HyperCard equivalent of a turtle step.) Experiment with some other numbers and see what effect they have.

Each electronic card in HyperCard is 512 pixels across and 342 pixels down. Therefore the coordinates 256,171 should place the compass directly in the center of the screen.

Logo & Company -- Continued



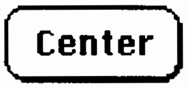
You can type commands in the message box which move the compass around this coordinate grid. It is also possible to create a button which will move the compass. Go to the Objects option on the menu bar, and select the New Button option to create a new button. Double-click on the new button to go to the Button Info dialog box. Within this dialog box, rename the new button "Center". Then click on the Script option within the Button Info dialog box. Selecting the Script option will take you to a new dialog box in which you can enter a script. ("Script" is the term used for a HyperCard program.)



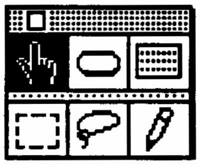
Script of card button id 2 = "Center"

```
on mouseUp
  set the loc of card button compass to 256,171
end mouseUp
```

Type the following command into script of card button "Center": "Set the loc of card button compass to 256,171". (The "on mouseUp" and "on mouseDown" parts of the script will already be present when the script box appears.) Then click OK. After you click "OK", you will return to the HyperCard screen, and the following button will be present.



Move the "Center" button to a convenient place on the screen. Then select the Browse tool from the tools palette. (The



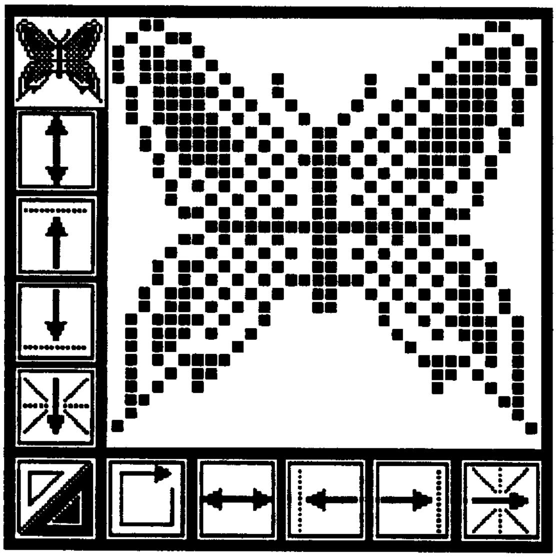
Browse tool is the one which looks like a hand.)

The Browse tool gives you a finger that you can use to press the buttons on the screen. Place the finger of the Browse tool in the middle of the Center button, and click once. Did the compass go to the center of the screen?

An important point: When the **Button** tool is selected, clicking the "Center" button has one effect, while another effect is produced when the "Center" button is clicked with the **Browse** tool. More specifically, double-clicking the "Center" button when the **Button** tool is selected takes you to the **Button Info** dialog box. On the other hand, clicking the "Center" button with the **Browse** tool will cause the script of the button (if any) to be run.

Now that you know how to make a button to control the position of the compass, make another button to set the compass to the top of the screen. The compass in this case acts something like a turtle. You can move it around the screen with HyperCard commands much as you can move the turtle around the screen with Logo commands. Some versions of Logo provide the user with multiple turtles. For example, LogoWriter provides up to four turtles, each of which can be a different shape. In HyperCard it is possible to have up to 32,000 different icons. There are a number of shareware and commercial icon editors which allow users to design their own shapes. One we particularly like is a commercial program called Icon Factory. As you can see from the butterfly on the next page, an icon editor allows HyperCard users to edit the shape of an icon in much the same way that a Logo shape editor allows Logo users to edit the shape of the turtle in some versions of Logo.

After you finish experimenting with different HyperCard buttons, select the **Go** option on the menu bar at the top of the screen. Then choose the **Go Home** option on the **Go** menu. This will take you back to the Home Card and save your work at the same time. From the Home Card you can select **Quit** under the **File** menu to exit HyperCard.



Summary

In this first column on parallels between Logo and HyperCard we have shown you how to do the following:

- •Start the HyperCard program
- •Create a new stack called "Demo"
- •Create a new button with the compass icon
- •Enter commands in the message box to control the position of the compass icon
- •Create a second button named "Center" and enter a script associated with the "Center" button which can be used to center the compass icon
- •Save the new stack by returning to the Home Card

Logo and HyperCard each have their own strengths, and each program can do some things the other can not. However, there are also similarities between the two programs.

Save your new Demo stack. We will use this stack as the starting point for next month's column. You probably felt that commands such as "Set the loc of card button compass to 50,100" were not as straightforward as FD 50 and RT 90. In next month's column we will consider how we can add some commands to HyperCard which are more "Logo-like". If you would like to continue working with HyperCard in the meantime, we recommend the HyperCard Handbook by Danny Goodman, published by Bantam Books. It is a good beginning introduction and reference manual for HyperCard.

Glen and Gina Bull
Curry School of Education, Ruffner Hall
University of Virginia, Charlottesville, VA 22903
Glen: LB2B@ VIRGINIA. Gina: RLBOP@VIRGINIA.

Logo: Search and Research

Research-based suggestions for understanding Logo

by Douglas H. Clements

Beyond facilitating debugging, what can be done to help students learn Logo with understanding? What aids them in forming complete and accurate representations for Logo programming concepts and processes? Suggestions are provided by research from both educational psychology and the classroom (Clements, 1989; Mayer, 1979). Most suggestions aim to improve conceptual understanding by having students elaborate on, and make connections between, the "bits" of knowledge they possess.

Learning commands completely

One important set of connections is that between Logo commands. One way to help students conceptualize such connections was provided in a previous column, "What's hard about beginning with Logo? The research." Students also need to elaborate on their knowledge of commands. They often learn only the bare minimum about each Logo command. They may see PRINT only as "a way to get one thing done"; in this case, as a way to print a word, number, or the like. They then do not see all the ways the command might be used. For instance, if they do not understand that PRINT also prints a carriage return (i.e., it moves the cursor down one line to the left margin), they will often not perceive PRINT (with an empty sentence or word as its input) as a way of double spacing, or creating a blank line. So, commands should be examined and discussed fully to be understood and applied effectively.

A worthwhile format for describing commands completely has been suggested by Harvey (1985). Modified slightly, it is:

Name the procedure;
State its purpose;
Tell whether it is a command or operation (reporter);
Tell about its inputs;
And tell what it does (if a command, what its effect is, if an operation, what it outputs).

For example, let's try it for FORWARD.

Name the procedure;	FORWARD
State its purpose;	is used to move the turtle forward, either to draw (with the pen down) or to change its position without drawing (with \the pen up).
Tell whether it is a command or operation (reporter);	It is a command
Tell about its inputs;	that takes one input, a number,
And tell what it does (if a command, what its effect is, if an operation, what it outputs)	and moves the turtle in the direction it is heading that number of turtle steps, leaving a path if the pen is down.

And for QUOTIENT:

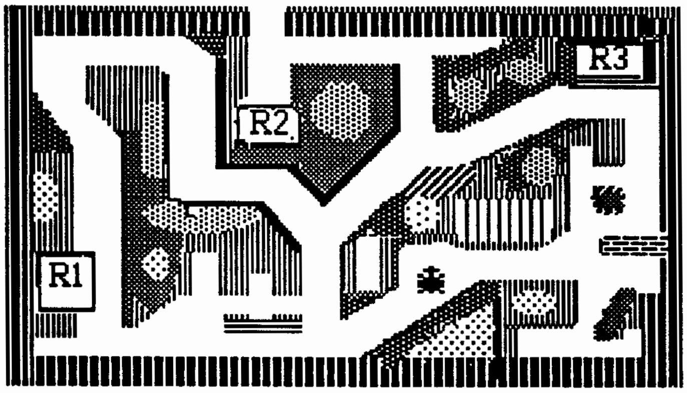
Name the procedure;	QUOTIENT
State its purpose;	is used to divide two numbers.
Tell whether it is a command operation (reporter);	It is an operation
Tell about its inputs;	that takes two inputs, both numbers,
And tell what it does (if a command, what its effect is, if an operation, what it outputs).	and outputs the results of dividing the first by the second.

Of course, having students memorize these definitions would most likely yield mere rote learning. Teachers should, however, find this format helpful for organizing essential information for themselves as well as for organizing meaningful learning experiences. Whether or not students can repeat this information, they should understand it to obtain full benefit from programming. Such elaboration helps "fill in the holes" in students' understanding. It can serve them as internal prompts, guiding them through programming problems.

Dramatization

Dramatizing programs is a powerful strategy for learning; one that should never be abandoned because students are (or think they are!) "too old." An illustration of this comes from a classroom using the Logo-based Geometry Curriculum Mike Battista and I have developed. Middle school students were working on the following task:

Write a procedure to teach the turtle to deliver a load of pies to the restaurant R1 and to return home. The path home from the restaurant must be the same as the path to the restaurant. Do the same for restaurants R2 and R3.



Students were working in a large group with the teacher using an overhead projection device that displayed the computer screen. As a group, they had given commands to move the turtle to R1. These commands were entered into the Logo editor. After some revisions, they were satisfied with the following procedure:

```
TO GO.R1
LEFT 130
FORWARD 40
RIGHT 40
FORWARD 85
RIGHT 90
FORWARD 30
LEFT 90
FORWARD 10
END
```

The next part of the task was to get the turtle to return to its starting position along the same path. After some discussion, Andy proposed the following:

Andy:	Everything that was a forward is a back ward, and everything that is a right is a left, and everything that is a left is a right, probably.
Teacher:	So you're telling me what?
Celia:	Reverse the stuff.
Andy:	You reverse, go from the bottom [of the procedure] to the top.
Teacher:	What comes first?
Celia:	BACK 10, RIGHT 90, ...

So the students are correctly giving the commands to undo the path. But though he proposed the solution, Andy is still concerned.

Andy: We've got a problem here, because here's the restaurant, it goes out of the restaurant BACK 10, then right would be up here so it's going to get screwed up.

Robin: No, now it's going backwards.

Andy did not understand Robin's reasoning. The students all start discussing Andy's misgiving about the proposed solution. They finally insist that Andy stand up and *act out* the commands. Doing so, he sees that you do need a right turn because, in the next command, the turtle is going backwards (he had envisioned the turtle going forward). The commands are then entered into the computer, and the students see that their proposed solution is correct.

Group dramatizations of recursive programs are especially helpful in ameliorating the problems with this process that we discussed two columns ago ("Recurrent Recursion Misconceptions"). Consider the procedure:

```
TO SQUARE
FORWARD 10
RIGHT 90
SQUARE
END
```

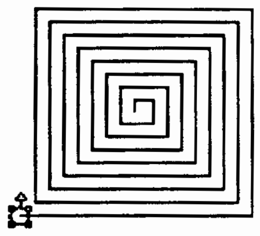
To dramatize this recursive procedure, one class had a boy be SQUARE. He told a second student acting as FORWARD to push the turtle forward 10 steps. FORWARD walked over to the turtle and pushed her forward. The turtle left a trail with tape. FORWARD then sat down. RIGHT was similarly instructed. Then the first student called SQUARE. This was not actually calling himself, but another copy of himself. (Some teachers have all "student-procedures" hold a copy of their code; in a recursive call, the calling procedures literally copy their code in creating a copy of themselves.) In other words, SQUARE called another student, a girl, who was defined exactly as SQUARE. This student stood up and called FORWARD and RIGHT, who sit down after doing their job. She then called another SQUARE. Notice, neither SQUARE has yet been told that everyone they called is done, so they cannot sit down. This continues as long as desired, or until the class runs out of students!

Search and Research – continued

Of course, this is merely an illustrative example. SQUARE would more likely be written without recursion. However, a modification makes some use of recursion:

```
TO SQUARE.SPIRAL :LENGTH.SIDE
FORWARD :LENGTH.SIDE
RIGHT 90
SQUARE.SPIRAL (:LENGTH.SIDE + 3)
END
```

In this case, the SQUARE.SPIRAL 10 procedure would still generate a copy of itself. But, instead of handing over 10 as the input to the new copy, it would first add 10 + 3, and hand over 13. Therefore, the instructions for the next copy of SQUARE.SPIRAL would be FORWARD 13 (the value of LENGTH.SIDE is for this copy of SQUARE.SPIRAL), RIGHT 90, and then, this SQUARE.SPIRAL would generate a copy of itself, handing over the value of 16. (You probably get the picture. The Logo picture you get—partially completed—is shown below).



Of course, SQUARE.SPIRAL does not have a stop rule, so it continues forever. We want it to stop when the length of the side is getting almost as long as the screen; that is, we need a conditional (IF/THEN) instruction.

```
TO SQUARE.SPIRAL :LENGTH.SIDE
IF :LENGTH.SIDE > 50 [STOP]
FORWARD :LENGTH.SIDE
RIGHT 90
SQUARE.SPIRAL (:LENGTH.SIDE + 3)
END
```

This new command tells each copy of SQUARE.SPIRAL first to check if the value they were given for LENGTH.SIDE is greater than 50. If so, that copy should stop running immediately after it informs the copy that called it that it's done. Then that copy can tell the copy that called it that it too is done, and so on and so on.

To return to the classroom dramatization: Each copy of SQUARE.SPIRAL would first check if the value of its LENGTH.SIDE were greater than 50. Say that a boy was the first to be given such a value (i.e., 52). He would have told the SQUARE.SPIRAL that called him (say a girl) that he was done and then sat down. She would have then told the SQUARE.SPIRAL that called her that she was done and sat down, and this domino-like process would have continued until everyone was sitting.

To test your understanding of the process, explain to yourself or someone else *exactly* how the class would dramatize this procedure:

```
TO SQUARE.SPIRAL.2 :LENGTH.SIDE
IF :LENGTH.SIDE > 50 [STOP]
FORWARD :LENGTH.SIDE
RIGHT 90
SQUARE.SPIRAL.2 (:LENGTH.SIDE + 3)
BACK :LENGTH.SIDE
RIGHT 90
END
```

Such dramatizations can help students build dynamic and accurate mental models of computer programs. They do this by helping students connect the abstract programming processes to actions that they make and therefore understand. The dramatizations also help students elaborate their knowledge of these processes, filling in levels of detail that may have otherwise been overlooked.

References

Clements, D. H. (1989). *Computers in elementary mathematics education*. Englewood Cliffs, NJ: Prentice-Hall.

Harvey, B. (1985). *Computer science Logo style: Intermediate programming*. Cambridge, MA: MIT Press.

Mayer, R. E. (1979). The psychology of how novices learn computer programming. *Computing Surveys, 13*, 121-141.

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

10th Annual International Contest

What is the International Computer Problem Solving Contest (ICPSC)?
To some schools it is a chance to challenge their top computer and mathematics students in an annual computer problem solving event that compares their solutions to the best in the world. To others it is challenging set of problems to be used as enrichment material for computer programming classes. However you choose to use the ICPSC, it can be a valuable resource for any computer programming teacher.

What is unique about the ICPSC?
The ICPSC combines the art of problem solving with the skill of computer programming. Our contest challenges teams (from one to three students each) to create short, original solutions to a set of five problems within a two-hour period. All problems can be solved with short computer programs written in any language on any computer system. The problems fall into five categories: 1) computation, 2) simulation, 3) graphic patterns, 4)words, and 5) mind benders.

Can an individual teacher in a school become a contest director for his/her school?
Yes! We want to challenge and enrich your students; you don't have to make it a major district-wide event if that would stop you from doing it. Of course the more students in your local area who can get involved, the better. But if you only have one student in your whole school who wants to compete, that is all you need.

Can Logo be used?
Our original contest problems were not appropriate for Logo students. So we added a new Logo contest, and last year we expanded this contest to include all three age groups.

What are all the age groups?
Elementary Division (Grades 4-6)
Junior Division (Grades 7-9)
Senior Division (Grades 10-12)
These are now offered in either the Open contest (all languages) or the Logo contest.

When is the contest held?
The 10th Annual ICPSC is scheduled for Saturday, April 28, 1990. Friday, April 27 and Monday, April 30, are alternate dates that can be used if Saturday is impossible. This year's set of contest problems will be mailed to the contest director on April 1, 1990. Sets of previous problems and solutions for students to practice on are always available.

How can I get more information about the ICPSC?
The Computing Teacher annually publishes articles about the contest in the November to February issues. Also the *Logo Exchange* is publishing stories on the Logo contest. You may receive a free copy of *Compute It!*, the official newsletter of the ICPSC, which is published in November each year, by writing to:

Dr. Donald T. Piele
ICPSC
P.O. Box 085664
Racine, WI 53408
Ph 414-634-0868 (Fridays)

Global Logo Comments

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

Logo Exchange Continental Editors

Africa	**Asia**	**Australia**	**Europe**	**Latin America**
Fatimata Seye Sylla	Marie Tada	Jeff Richardson	Harry Pinxteren	Jose Valente
UNESCO/BREDA	St. Mary's Int. School	School of Education	Logo Centrum Nederland	NIED
BP 3311, Dakar	6-19, Seta 1-chome	GIAE	P.O. Box 1408	UNICAMP
Senegal, West Africa	Setagaya-ku	Switchback Road	BK Nijmegen 6501	13082 Campinas
	Tokyo 158, Japan	Churchill 3842	Netherlands	Sao Paulo, Brazil
		Australia		

This month's international column brings *Logo Exchange* readers up-to-date on what is happening in Australia. As 1990 has now reached us, many *LX* readers may wish to begin planning to attend the World Conference on Computers in Education in Sydney this coming July — more about this later in the column. Here is Jeff Richardson's report.

Brian Silverman visits Australia

Brian Silverman recently visited Australia very briefly as a guest speaker at the annual conference of the Computer Education Group of Victoria. Brian's keynote address, built around a demonstration of his "Phantom Fishtank," was provocative and very well received by a mixed (i.e., not 100% Logophile) audience. However, Brian managed to top this the next day when he conducted a computers-and-mathematics workshop and proceeded to amaze the participants with an exploration of non-linear dynamics of integers - in LogoWriter!!! In between times he was besieged by LogoWriter users who variously expressed their thanks, asked for tips and background stories, or presented him with their shopping lists of features they'd like to see in the next upgrade. Brian bore up will, but I hope we haven't frightened him off returning for the World Conference in Sydney in July.

Turtle Confusion* and *Turtles Speak Mathematics

If you enjoyed the pictures beamed back to us from Jupiter by the Voyager last year, then one of the many people you have to thank is Barry Newell. Barry, an astrophysicist, is the administrator of the Mount Stromlo and Siding Spring Observatories outside of Canberra in the Australian Capital Territory. Barry is also the author of two new Logo books *Turtle Confusion* and *Turtle Speak Mathematics* (published through the Curriculum Development Centre, P.O. Box 34, Woden, ACT 2066, Australia). Neither book even remotely resembles anything you've ever read before, except perhaps Lewis Carrol's *Alice in Wonderland* stories. Like the Reverend Dodgson (Lewis Carrol's real name), Barry has love and respect for both children and mathematics. This shines through in both of these enigmatic, challenging and amusing books, both are a must to add to your Logo library.

Australian National Conference on Computers in Education

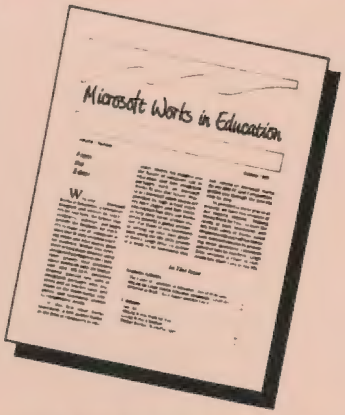
The Australian National Conference on Computers in Education was held in Canberra in the Spring of 1989. There were several Logo related presentations. Pam Gibbons of the Catholic College of Education in Sydney outlined the joys and sorrows she shared with her undergraduate class as they combined Logo and journal writing in her innovative problem solving course. Dr. Anne McDougall of Monash University silenced the skeptics with her demonstration of a very young child's ability to use embedded recursion by writing a Logo version of Dr. Seuss's *The Cat in the Hat*. Peter Carter (author of *Thinking Logo*) set loose a menagerie of walking machines, Lego robots built and programmed in LogoWriter by his tenth grade class. Barry Newell (see above) was a keynote speaker. He set the conference back on its heels with a confident and lucid reassertion of the role of Logo programming in helping children to use symbolism and modeling to build understandings across the curriculum.

WCCE 90

Australians are looking forward to meeting as many Logophiles as can make it to WCCE 90, the Fifth World Conference on Computers in Education, in Sydney, Australia in July 1990. Seymour Papert, Andrea di Sessa and Alan Kay are among the invited speakers and there will be Logo and Logo related sessions across all five streams of the conference. Of special interest to *Logo Exchange* readers should be the opportunity to visit Australian schools before and after the conference. Direct any enquiries to Dr. Wing Au, School of Education, University of Newcastle, 2308, Australia.

Microsoft Works in Education: Helpful homework.

When you bring a copy of *Microsoft Works in Education* home, help is in your hands.



Microsoft Works in Education is a 24-page newsletter published four times a year. Its goal: to help you, the K-12 computer using educator, simplify your job by sharing tips, experiences, and success stories of other teachers who use MS *Works* in the classroom.

Whether you're looking to customize your gradebooks or increase your students' knowledge of the world through using databases, *Microsoft Works in Education* supplies ideas and information to get you started.

If Microsoft *Works* is a part of their homework assignment, let *Microsoft Works in Education* help you with yours.

Subscription Rates:

	ISTE Member	**Non-Member**
U.S.	$12.00	$14.50
Non-U.S.	$15.00	$17.50

ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; ph. 503/686-4414.

ef•fec•tive\i-'fek-tiv\adj (14c)

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

Computer-Integrated Instruction: Effective Inservice

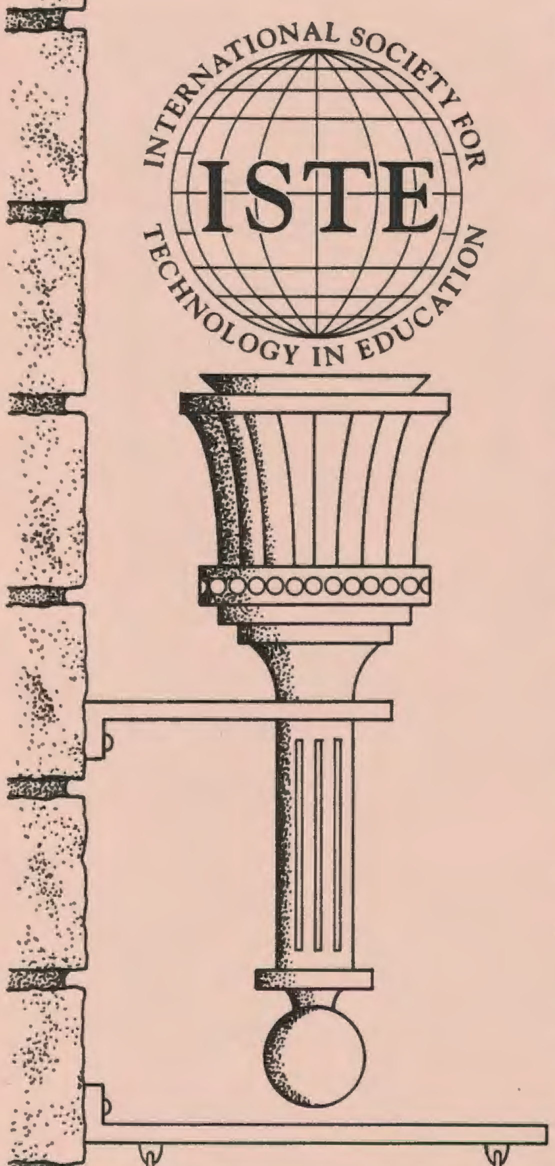
Dave Moursund's comprehensive series on inservice training for computer using educators has grown. ***Effective Inservice for Secondary School Mathematics Teachers*** and ***Elementary School Teachers*** are joined by texts for ***Secondary School Science Teachers*** and ***Secondary School Social Studies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

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

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

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

ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; ph. 503/686-4414.



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

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

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

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

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

Rely on ISTE support:

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

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

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

ISTE, University of Oregon,
1787 Agate St., Eugene, OR 97403-9905.
ph. 503/686-4414.