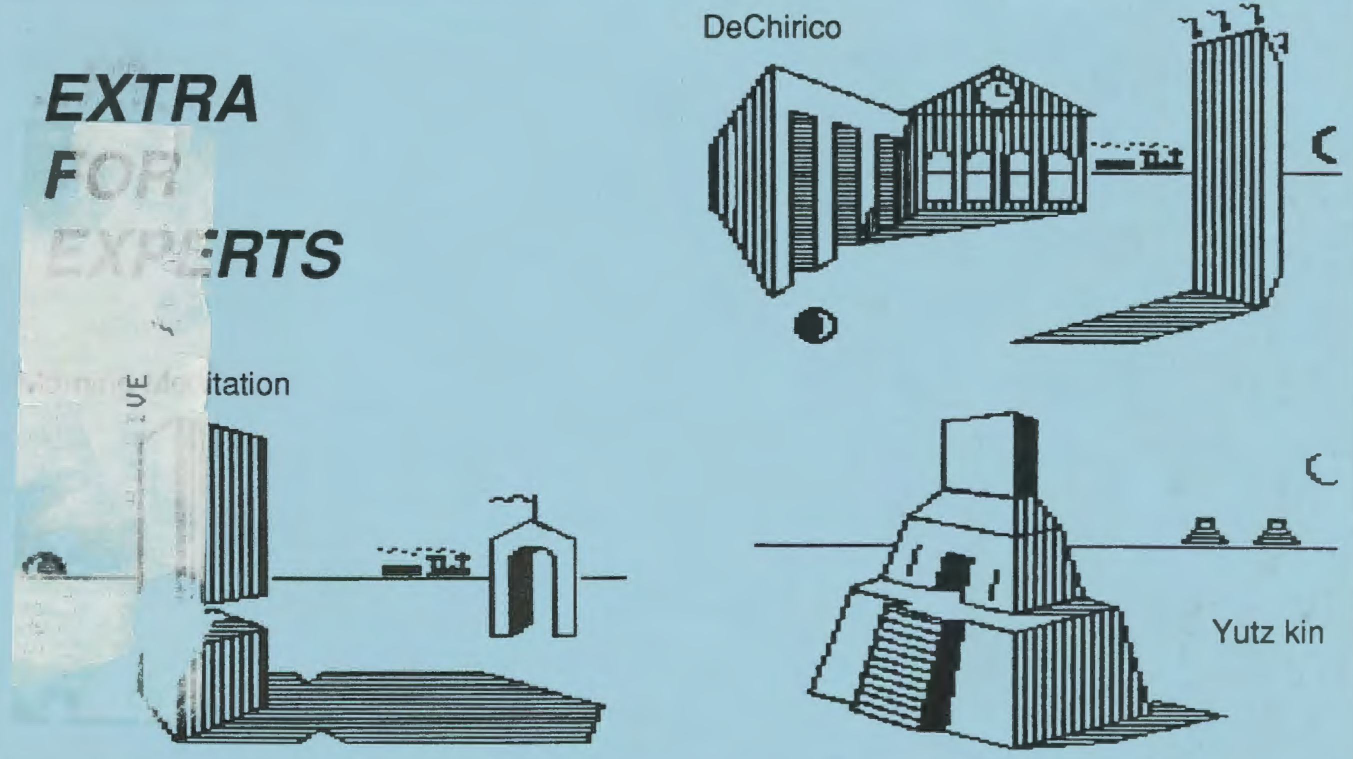
Journal of the ISTE Special Interest Group for Logo-Using Educators

LOGO EXCHANGE

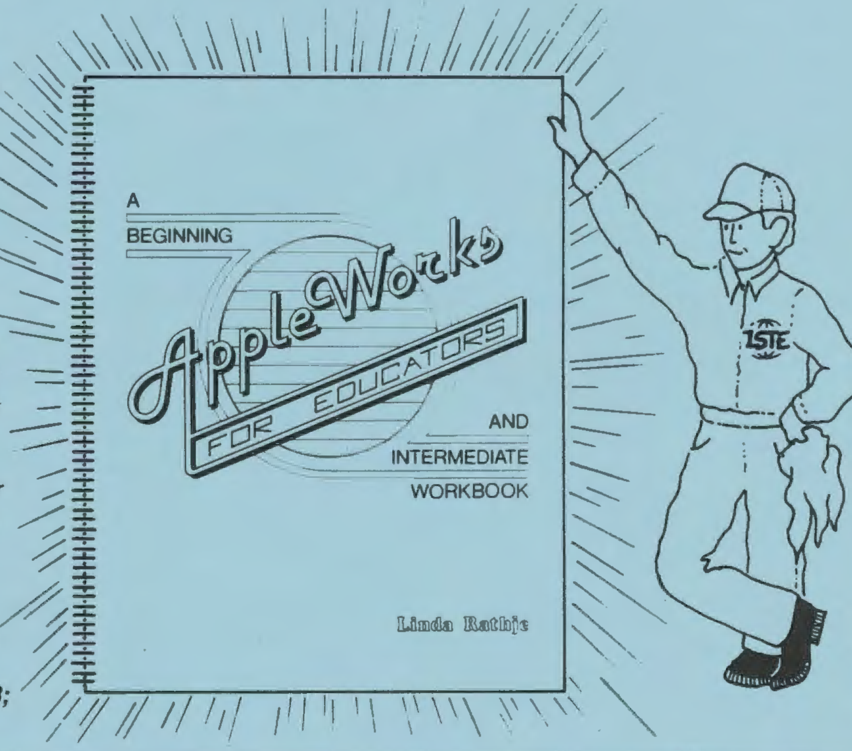
April 1990 Volume 8 Number 8

EXTRA FOR EXPERTS



International Society for Technology in Education

ISTE Publications


A
BEGINNING
AppleWorks
FOR EDUCATORS
AND
INTERMEDIATE
WORKBOOK
Linda Rathje
ISTE

LOGO EXCHANGE

Volume 8 Number 8 Journal of the ISTE Special Interest Group for Logo-Using Educators April 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; 503/346-4414. This publication was produced using Aldus *PageMaker*®.

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

Contents

From the Editor—So, what *is* a "Logo Expert"?
Sharon Yoder 2

Monthly Musings— Re: Pete and Curses
Tom Lough 3

Logo Ideas—Who's the Expert? Levels of Expertise
Eadie Adamson 4

Beginner's Corner—April Fools!
Dorothy Fitch 7

Implementing Powerful Ideas—the Case for RUN
Rina Zazkis and Uri Leron 11

Logo LinX—Reach Out and Touch Logo
Judi Harris 15

MathWorlds—The Case of the Babylonian Turtle: A Logo Mystery
Jim King
Sandy Dawson, editor 18

Three-Dimensional Logo
Horacio C. Reggini 21

Logo & Company—Building a Turtle in HyperCard: "Mock Turtle"
Glen Bull, Gina Bull 27

Global Logo Comments—Notes From Around the World
Dennis Harper, editor 31

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

So, what *is* a "Logo Expert"?

The theme of this issue of the *Logo Exchange* is "Extra for Experts." While it may seem obvious that our intention is to provide some "advanced" material for our more sophisticated readers, the question still remains. Do we *really* know what we mean when we say "Oh, *she's* the Logo expert around here!"

It seems to me that there are several possible interpretations for the phrase "Logo Expert." We might mean the person who is a master of the Logo programming language. Perhaps you've met such a person. No matter what your question about the Logo language, these people always seem to have the answer. They know all of the seldom-used commands and subtle eccentricities of your favorite version of Logo. Often these experts are knowledgeable about many versions of Logo. "Oh," they comment, "in LogoWriter you use OR, but in Logo PLUS you use ANYOF." It's amazing the details these experts carry in their heads.

Another category of Logo Expert that we value is the talented debugger. This is the person you find when your program simply won't run correctly. These people take one look at your procedures and often immediately see the source of your difficulties. If they don't have an instant answer, then they seem to know exactly where to look or how to locate a solution for your problem. These folks have a genuine gift for problem solving in the Logo environment.

But being a Logo Expert can mean more than just having skill as a programmer. We often think of the person with a broad knowledge of the Logo philosophy as being a Logo Expert. Such experts may not be masterful programmers, but they have a deep understanding of what is meant by the "Logo environment" and know how to develop such environments in the classroom. These are the teachers we love to watch work; these are the classrooms rich in discovery learning in the fullest of Logo traditions.

The *Logo Exchange* itself is full of columns and articles by a variety of Logo Experts. If you read these pages regularly, then you know that Tom Lough is an expert at finding Logo in all aspects of life, while Eadie Adamson has a real gift for challenging her students to learn more Logo in the midst of fascinating projects. This year, Dorothy Fitch has been our expert at providing ideas for beginners. Doug Clements is our expert on research, while Dennis Harper is an expert on the uses of Logo world-wide. Sandy Dawson brings us articles from a variety of Logo-and-math experts, while Judi Harris shares her special talents with language, often being our Logo-and-language-arts expert. This year Glen and Gina Bull have focused their expertise on Logo connections and Logo-like ideas. Each of these people shares a bit of his or her expertise with us all each month.

In thinking about this idea of experts, I am reminded of a conversation that occurred nearly 10 years ago. I was just beginning to use a computer in my classroom and was being called an "expert" by colleagues and administrators. I certainly didn't consider myself an expert at all. I expressed my reservations to my friend, whose work in the area of computer education I had admired for many years. He responded by reminding me that I knew more about computing than anyone else in my district and so was indeed the expert as far as they were concerned. That reassurance by someone I respected gave me the courage to go back to my district and eventually build a Logo-rich K - 12 computer education program of which I could be proud.

So, if you are being called "Logo Expert" by your colleagues, take it as a compliment. Examine your knowledge carefully. In what areas are you *really* an expert? Have confidence in yourself. Continue to learn and grow in your own special area. Broaden your horizons whenever you can. And, most importantly, share your expertise with your colleagues.

Sharon Yoder
ISTE/SIGLogo
1787 Agate Street
Eugene, OR 97403
Ph: 346-4414
CIS: 73007,1645 BITNET: YODER@OREGON

Monthly Musing

Re: Pete and Curses

by Tom Lough

My six-year-old told this story at the dinner table recently.

"There once was a boy named Pete and his brother, Repete. They were walking down the road and Pete ran away. Who was left?"
"Repete."
"There once was a boy named Pete and . . ."

After a couple of "repete-titions" of this, I managed to pull a Control-S (or Apple-S) and turn his attention to the matter of dinner. But this little story gave me some, er, food for thought. In the March Musing, I commented on some different ways to look at the REPEAT command. My son's story prodded me to continue. Of course, my first thought was something like the following.

```
TO REPETE
PRINT [There once was a boy named Pete and
    his brother, Repete.]
PRINT [They were walking down the road and
    Pete ran away. ]
PRINT [ Who was left?]
REPETE
END
```

I am always on the lookout for effective ways to help Logo learners make the transition from repetition to recursion, and to be able to tell the difference between the two processes. Unfortunately, this distinction is often not very clear.

A typical learning sequence goes from REPEAT into some form of tail recursion. (Someone might ask, "How can I get the turtle to keep doing this forever?")

```
TO MOVEIT.1                    TO MOVEIT.2
REPEAT 100 [FORWARD 5          FORWARD 5
RIGHT 5]                       RIGHT 5
END                            MOVEIT.2
                               END
```

Sometimes learners will stumble upon a clue that there is a little more to recursion than simple repetition.

```
TO SPIRAL.1 :SIZE              TO SPIRAL.2 :SIZE
IF :SIZE < 10 [STOP]           IF :SIZE < 10 [STOP]
FORWARD :SIZE                  SPIRAL.2 :SIZE - 10
RIGHT 90                       FORWARD :SIZE
SPIRAL.1 :SIZE - 10            RIGHT 90
END                            END
```

How can we help learners construct a mental picture of what is going on? There are several suggestions in the many recursion articles that have appeared both in the *LX* and in *The Computing Teacher*. But I feel we need descriptions that are based on more ordinary experiences. Here are a couple for consideration, based on activities that many perform in an otherwise unremarkable manner.

1. When you are reading a difficult book and you come upon an unfamiliar word, you put a bookmark where you are reading and turn to another section to read about the unfamiliar word. During that reading, you might come upon a second unfamiliar word, put a bookmark there, and turn to yet another section... Sooner or later, you find something you understand and go back to your most recent bookmark to resume. When you finish that, you go to the next most recent bookmark, . . . and finally, you are back where you started, and resume what you originally set out to read. The important idea is to recognize the sequence of bookmarks that waited for you to back out through them in turn.

2. I have been struck by the ease with which we shift into and out of different levels or subjects as we talk with each other. If you listen closely to ordinary conversation, you might hear something like the following. Notice how the story line starts with one topic, pauses with a conversational bookmark, goes to another topic, and then comes back to where it was.

"I went fishing with James and Penny last week. That reminds me, the boating show is coming up. Want to go? It is sure to be even bigger than last year. Anyhow, there we were, out on the lake, not a bite all day, when suddenly, . . ."

Thinking about activities such as these may help learners to understand recursive procedures such as the following, and may stimulate them to write recursive procedures of their own, when appropriate.

```
TO ADD.LIST  :LIST
IFELSE NOT EMPTY? :LIST [ OUTPUT (FIRST :LIST)
    + ADD.LIST BUTFIRST :LIST ] [ OUTPUT 0 ]
END
```

I believe that recursion is one of the most powerful processes within Logo. Let's all be on the lookout for additional everyday experiences that contain some of the important elements of the recursive process, and share them with our favorite Logo learners—including Pete and Repete, of course!

FD 100!

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

Logo Ideas

Who's the Expert? Levels of Expertise

by Eadie Adamson

Herewith, some thoughts on experts.

How often do you use the word "expert" in a Logo class? I use it often.

"How do you fill?" "Ask David, he's an expert."
"What's wrong here?" "Ask Philip, he's an expert at debugging things like that."
"How do I....?" "Check with ..., he's an expert."

I like to think that everyone is an expert at something, so this column for "experts" will be somewhat philosophical as well as procedural. It is important that everyone feel the special kind of confidence that comes from being pointed out as "expert" in some way.

There's another issue about being "experts." Learning to use LogoWriter, or any other version of Logo, most people follow a fairly predictable path that was characterized wonderfully by Brian Harvey in a talk he gave at the fall CUE conference last October. Brian spoke of programming as an art. He broke learning to program into three stages:

1. Learning the rules.
2. Apprenticeship projects.
3. Computer science.

Brian talked of an aspiring artist (read: programmer) and what that artist must learn. Brian was talking about high school students. But I think that what he had to say about the process of learning to program and the stages of this development relates directly to what we see happening with teachers and students in classrooms, whether in elementary school, high school, or colleges. First, a student must gain the technical knowledge. To use LogoWriter, everyone must learn the language. That takes some time, more time for some than for others. Once past learning the language (more like learning to skate or ride a bike), there is the new-found ability to play with the language, explore it, do some simple and perhaps not so simple things with it. This period of time, working on developing the technical skill, is the period of practice, and it can last a very long time.

When we are beginning to learn to use Logo, too often we confuse Brian's step 1 and step 2. Often we seem to leave out step 1 altogether. We need to pause and realize that our students need the chance to learn the rules first and that this takes time. Once the rules are learned, Logo projects can and will flourish. Like a skater or bike rider, the basic skill is essential to that wonderful free movement. Successful uses of Logo in the classroom depend upon having students who have learned enough of the rules to be able to begin to work on projects that they have initiated. This is the time, as Brian characterizes it, in which the teacher becomes the resource. Learners turn to the teacher as the how-to person when a great idea just won't work because their skill level has not reached the appropriate height.

Michael Tempel and I have talked about this idea also, relating it to how a leader can make clear in teacher training sessions what the expectations should be. First, participants need to have a chance to learn the language. Then, once this is well begun, they can begin to consider applications. Yet all too often people come to workshops with scanty knowledge and expect to leave with a "recipe" for curriculum applications under their belt, a kind of magical infusion that will just "happen." Teachers need a chance to learn *for themselves* first. Only then can they begin to think clearly about ways to use Logo. We have recently broken our workshops into stages, hoping to make this approach clear and also hoping that by devoting time to stage 1 (learning the language), stage 2 learners will be even more successful. Learn the language first, for yourself, and then learn how to use it. The same goes in the classroom.

Since we're on the subject of language.....

Recently I've had the opportunity to work with a colleague's fifth grade English class and to try some variations on some of the language play so wonderfully outlined in Paul Goldenberg and Wallace Feurzeig's *Exploring Language with Logo*. These boys had a good grasp of rudimentary Logo. It was the right time to take them beyond the problems of programming and show them how they might use Logo to help them think about language.

We are all experts (that word again!) on our own natural language, but sometimes we need a new experience, for instance work with Logo and language by generating sentences, to discover just how much we really know.

Now, in this class we had a collection of "experts": I was the "Logo expert;" Jennifer was the "grammar expert, alias teacher;" the students were all "English experts," but they didn't know it yet. They were also, at another level, "Logo experts" too. As experts we each had a job: Since I was the Logo expert, I provided the beginnings of procedures for us to play with. Jennifer was the grammar expert, so she kept me out of trouble when it came to using grammatical terms that the boys had been studying. The boys were the language experts, or critics, who would not only supply the words but decide whether the sentences we generated were correct English. Eventually, as budding Logo experts, they would begin to write their own procedures and create their own language masterworks.

Jennifer and I wanted to see if a bit of language exploration with Logo would help the boys begin to use more descriptive words in their writing. We wanted to take the thinking about the words and place it in another context. Moving to the world of Logo, rather than focusing on their compositions, the boys might be able to explore more freely and might eventually carry some of that imaginative play with words back to their writing. I was also delighted to have an opportunity to work with Logo and language outside of the context of a regular Logo class, since my classes usually get so involved with projects that it is very unlikely I could get everyone's full attention (willingly, at least) for such a project.

We began by working with a single computer and a large monitor. I started LogoWriter with one procedure already installed, that **pick** procedure that really ought to be a Logo primitive. (I find I use it so often in so many situations that, although I can add it quickly as a tool, it would be much nicer if it were a primitive. Are you listening: LSCI? Terrapin?)

```
to pick :list
output item 1 + random count :list :list
end
```

Jennifer and I had thought to begin with noun phrases and verb phrases, but the boys immediately wanted to generate simple noun-verb sentences. Master plan number one went out the window; noun and verb phrases could come later! We created a collection of nouns after I explained how the noun procedure worked. (They were only slightly familiar with **output**, not at all with the **pick** procedure). Our unedited beginning for a noun—the nouns get added within the brackets:

```
to noun
output pick [    ]
end
```

On to the verb! It turned out everyone was a verb expert. Here was the framework for the verb:

```
to verb
output pick [  ]
end
```

Now we needed to try printing a sentence. Aha! another Logo word as well as part of English grammar. What an interesting fit!

```
print ( sentence noun verb )
```

I introduced the parentheses here so that it would be easy to add sentence parts without adjusting our process. Sentence, you experts will recall, takes only two inputs unless sentence and its inputs are surrounded by parentheses. (Yes, I know you can surround print with parentheses and print the whole thing without the word sentence, but we're working on English grammar here, remember? The sentence is what we want as output!)

We tried a few sentences, then sat back and scanned the monitor, critiquing what we had produced.

(By the way, occasionally you will see versions of language play that create variables for a noun and verb, as in **make "nouns [ ]** and **make "verbs [ ]**. As an "expert" you might see that this creates quite a bit of clutter in the computer's memory. Think of all the names to store! Writing a procedure to **output** the words works just as well without cluttering up the space. For programming style, it's more elegant and ... more expert(?) to **output** something rather than create global names. A nice general rule might be to use **output** when you can; otherwise create names using **make** or **name**.)

It was not until we had generated a few noun-verb sentences that the boys' English expertise began to show up. "We need an article," said one. Quickly the group supplied a few articles for a procedure we named "art" and we tried again.

Now we needed adjectives too, as the ideas for sentence form grew more complicated. Eventually we concluded that the words we chose needed to have some logical relationships for the sentences to make any sense. The boys also saw they needed to distinguish between animate and inanimate nouns, for instance, for active verbs to make much sense in their sentences. We added a new procedure, **person**, and included everyone's name. We connected a sentence to another with "BUT. I asked for a substitute for "BUT and someone came up with [AT THE SAME TIME] which created some interesting situations. Periodically we stopped to print out the sentences generated before clearing the screen.

Since then, the boys have begun to work on their own, newly "expert" at creating procedures to output a part of speech, and with some help from a handout I gave them for guidance. Yesterday I peeked into the classroom and found two boys at work at one of their classroom computers, trying to add punctuation to their sentences! Wonders never cease! I gave them a little help on that one. Their teacher, by the way, was delighted. It seems we've created a problem, though. Now decisions have to be made: who has priority on the computers (they have two) – the boys who want to write or the boys who want to play with language. I retreated quietly to my lab!

Bibliography
Goldenberg, E. Paul and Feurzeig, Wallace. (1987) *Exploring language with Logo.* Cambridge, MA: MIT Press.

P.S.: My thanks to Jennifer Jahos and her fifth grade class for all their enthusiasm as we continue to work together on language!

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

Creating Sentence Patterns

Below is an essential tool.procedure. Either put it on the LogoWriter page you are working with now or make a tool page and add a procedure to your page to load this as a tool using **gettools** ***"name.of.tool.page.***

```
to pick :list
output item (1 + random count :list) :list
end
```

Make lists of words that are different parts of speech. Make a list of article, adjectives, nouns, verbs, etc.

Put the words in the brackets in the procedures below.

If you have a two-word part of speech—"pea green," for example—include it in its own set of brackets *within* the other brackets. Example: [[pea green] red blue]

```
to article
output pick [             ]
end

to adj
output pick [             ]
end

to noun
output pick [              ]
end

to verb
output pick [              ]
end

to adv
output pick [               ]
end
```

Using only the names of the parts of speech, make your own patterns:

```
print (sentence ____   _____   _____ )
```

What combination makes a good descriptive sentence?

What makes a headline?

How about slogans, poems, movie titles?

How might you need to change your lists to produce good results?

Can you write a procedure to produce multiple sentences on the page?

An ISTE Copy Me! Page ***Developed by Eadie Adamson***

Just for Beginners

April Fools!
by Dorothy Fitch

This issue is supposed to be for experts. So what could a beginner's column possibly offer? Here are a few tips and tricks that will make you look clever, make your students think twice about what they see on the screen and on paper, and give you some tools to create expert-style programs for your students.

April Fools Puzzlers
(A couple of silly ideas to confuse and bewilder your students on April 1)

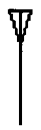
From the "What's Wrong with the Turtle?" Department

With a shape editor, you can create a new shape that looks just like the turtle. In Logo PLUS you can even create a turtle that points toward the bottom of the screen when you type HOME or DRAW. Imagine the look on your students' faces when they type FORWARD 50 and the turtle goes backwards, still facing down. If they type RIGHT 180, it will turn to point straight up, but when they type FORWARD 50 again, it still moves backwards.

Here's how to create this baffling turtle.

1. Load Logo PLUS.
2. Type DRAW to see the standard turtle.
3. Type RIGHT 180 to point it in the opposite direction.
4. Type STAMP to stamp its image in the center of the screen.
5. Type EDSHAPE 0 1 to enter the shape editor with the contents of the screen image.
6. Press Control-C to leave the shape editor and define the image in the shape editor.

Here's your new turtle shape. It looks like the regular turtle except that it points towards the bottom of the screen. When you experiment with it you will find that the movement commands are reversed, but that turn commands work normally. That is, a right turn is still a right turn.


FORWARD 50

Type SETSHAPE 0 if you want to return to the normal turtle. Type SETSHAPE 1 to use the upside-down turtle.

To save your new shape, type SAVESHAPES "AFTURTLE (for April Fools Turtle). To use it with your students, load Logo PLUS, then type READSHAPES "AFTURTLE. Then type SETSHAPE 1 to set the turtle to its new shape. (Do all this before your students arrive so that they won't know that a shapes file has been loaded.)

You'll enjoy watching your students as they try to figure out what has happened. Challenge them to draw a design with this upside-down turtle. Discuss with them how you created the shape and why it works the way it does. Let them create new shapes for the turtle too!

From the "Logo Has Gone Crazy!" Department

When you run the following AF procedure, it will look like you've just loaded Logo. There is even a Logo question mark prompt sign. However, when students type a Logo command, they will get an unexpected message. This is just a sample; of course you can tailor the messages for your class. The comments in the procedures (shown in small type) will help you understand how they work.

```
TO AF
NODRAW
    ; clears the screen and displays the Logo PLUS greeting
PRINT [Logo PLUS by Terrapin]
PRINT [====================]
PRINT [ ]
PRINT [(c) 1981 MIT]
PRINT [(c) 1989 Terrapin, Inc.]
PRINT [Version 1.1 - 128K (ProDOS)]
    ; use your version number, if different
PRINT [ ]
PRINT [WELCOME TO LOGO PLUS!]
PRINT [ ]
GET.COMMAND
    ; call to the GET.COMMAND procedure
END

TO GET.COMMAND
PRINT1 "?
    ; puts a Logo-type prompt sign on the screen
MAKE "COMMAND UPPERCASE REQUEST
    ;gets the user's command and names it
    "COMMAND UPPERCASE converts lower
    case commands to upper case in Logo
    PLUS
DO :COMMAND
    ; calls the DO procedure, giving it the command that
    was entered (:COMMAND)
GET.COMMAND
    ; a recursive call to GET.COMMAND so that the next
```

```
     command can be "parsed"
END

TO DO :COM
IF EMPTY? :COM THEN STOP
     ; prevents Return from stopping the procedure and
     makes it look like Logo since REQUEST (in the
     GET.COMMAND procedure) returns the user's input in
     the form of a list, you must test for commands in list
     form (in square brackets).
IF :COM = [DRAW] THEN DRAW PRINT [I CAN'T
     THINK OF ANYTHING TO DRAW.] STOP
     ; these next four lines print a message when particular
     commands are typed
IF :COM = [HOME] THEN HOME PRINT [SWEET
     HOME!] STOP
IF :COM = [POTS] THEN PRINT [AND PANS.]
     STOP
IF :COM = [CATALOG] THEN PRINT [SEARS OR
     MONTGOMERY WARD?] STOP
IF :COM = [FILL] THEN PRINT [I'M ALREADY
     FULL!] STOP
     ; To test just the first word that is typed, use FIRST
     :COM, which returns the first item in the command list
     that is entered. For example, FORWARD or BACK
     with any number moves the turtle a random amount
     between 0 and 99. RT or LT turns the turtle a random
     amount between 0 and 359 degrees.
IF FIRST :COM = "FD THEN FORWARD RANDOM
     100 STOP
IF FIRST :COM = "BK THEN BACK RANDOM 100
     STOP
IF FIRST :COM = "RT THEN RIGHT RANDOM 360
     STOP
IF FIRST :COM = "LT THEN LEFT RANDOM 360
     STOP
     ; These instructions make Logo do the opposite of the
     command that is entered.
IF FIRST :COM = "ST THEN HT STOP
IF FIRST :COM = "HT THEN ST STOP
IF FIRST :COM = "PU THEN PD STOP
IF FIRST :COM = "PD THEN PU STOP
     ; Giving a command to change the pen or background
     color produces an interesting message or display.
IF FIRST :COM = "PC THEN PRINT [I'M OUT
     OF INK!] STOP
IF FIRST :COM = "BG THEN BG 1 BG 2 BG 3
     BG 4 BG 5 BG 0 PRINT [AWESOME!] STOP
     ; If a REPEAT command is given, the instructions in
     the list are printed on the screen as many times as they
     should have been executed.
IF FIRST :COM = "REPEAT THEN REPEAT
     ITEM 2 :COM [PRINT LAST :COM] STOP
     ; Finally, this command prints HUH? (or some
     message) for any other untested command that is typed.
PRINT [HUH?]
     ; or [JE NE COMPRENDS PAS!]
END
```

Add additional tests to the DO procedure for other commands with which your students are familiar. To start the fun, type AF. (Do this before students arrive so they think that you have just loaded Logo. Tell them they can have some free time to explore Logo.)

After your students have had some fun with this program let them change the messages and add other instructions to the DO procedure, using existing lines as models. Then they can switch computers and see what their friends have created.

For older students, you can remove some of the silly messages and instead add some more thought-provoking effects.

For example, this instruction moves the turtle 10 times the input given:

```
IF FIRST :COM = "FD THEN FORWARD
     (LAST :COM) * 10
```

This instruction turns the turtle half the amount of the student's command:

```
IF FIRST :COM = "RT THEN RIGHT
     (LAST :COM )/ 2
```

This instruction moves the turtle backwards the distance given in a FORWARD command:

```
IF FIRST :COM = "FD THEN BACK (LAST :COM)
```

See if your students can figure out the special function for each command.

Are You Sure?

Here is a challenge that I have used with both children and adults learning Logo. They only need to have a little experience with turtle graphics commands and simple procedure writing.

Here is the task: Draw a picture of what would be drawn on the Logo screen if you type BARBELL.

```
TO BARBELL          TO CIRCLE
CLEARSCREEN         REPEAT 4 [FORWARD 20
                    RIGHT 90]
CIRCLE              END
LEFT 90
FORWARD 90
LEFT 90
CIRCLE
END
```

You may be surprised at the variety of designs you get! More people than you would imagine will pay more attention to the names of procedures than to the actual instructions in the procedures. It just goes to show you how important it is to give a little thought to the names you give your procedures. Type these instructions in Logo to see the correct answer!

Keyboard Fun

You may be familiar with the READCHARACTER command in Logo, which allows you to access keys on the computer's keyboard. (You may want to consult your Logo documentation for complete details on using this command.) "Instant"-type single keystroke programs typically use READCHARACTER to let young learners press letters to make the turtle move and turn. For example, they can press F to move the turtle forward 10 steps, R to turn it to the right 30 degrees, and so on. But did you know that you can also access non-letter keys, Control characters and Open-Apple keystrokes?

Each key on the keyboard has a special number code assigned to it, which is how the computer distinguishes among the keys that you press. This number is known as its ASCII code (American Standard Code for Information Interchange). You can "look up" the ASCII code for a particular key by typing this instruction:

```
PRINT ASCII READCHARACTER
```

or

```
PR ASCII RC
```

Press Return and then press any key. The number that you see is the ASCII code for that key. The next procedure, lets you look up as many ASCII codes as you wish. (Press Control-G to stop this procedure; its ASCII code, which is 7, won't be printed, since Control-G is a special Logo keystoke.)

```
TO CODES
PRINT ASCII READCHARACTER
CODES
END
```

You will notice that upper case letters have a different ASCII code than lower case letters, punctuation marks each have their own ASCII code, and special keys (Esc, Delete, Return, Tab, space bar, arrow keys, etc.) have a unique number. The Control key doesn't have an ASCII code of its own, but produces one when used in combination with letter keys. The ASCII code for Control-A is 1, for Control-B is 2, for Control-C is 3, etc. If you know the ASCII code for a character, then you can print it by typing an instruction like this:

```
PRINT CHAR 65
A
```

The Open-Apple key is equivalent to Paddlebutton 0, and both the Closed-Apple key and the Option key are equivalent to Paddlebutton 1. The paddlebuttons don't have ASCII codes, but they can be detected. Try typing

```
PRINT PADDLEBUTTON 0
```

Logo responds FALSE. But if you type the instruction again and hold down the Open-Apple key when you press Return, Logo will respond TRUE. The Logo command PADDLEBUTTON returns TRUE if the Apple key is being pressed and FALSE if not. Try the same instruction with PADDLEBUTTON 1 and the Closed-Apple or Option key on your keyboard.

Here is a sample set of procedures to demonstrate how to use of all these special keys, including the Apple keys:

```
TO INSTANT
DO.KEY READCHARACTER
INSTANT
END

TO DO.KEY :KEY
IF (UPPERCASE :KEY) = "F THEN FORWARD 10
   STOP
IF (UPPERCASE :KEY) = "R THEN RIGHT 30
   STOP
IF :KEY = CHAR 32 THEN PRINT [THAT'S THE
   SPACE BAR.] STOP
IF :KEY = CHAR 13 THEN PRINT [THAT'S THE
   RETURN KEY.] STOP
IF :KEY = CHAR 9 THEN REPEAT 5 [PRINT1
   CHAR 32] PRINT [TAB] STOP
   ; prints 5 spaces and then the word Tab.
```

```
IF ALLOF (PADDLEBUTTON 0) (:KEY = CHAR 8)
   THEN SETX XCOR - 10 STOP
   ; moves the turtle 10 steps to the left if the left arrow is
   pressed while the Open-Apple key is pressed down.
   (ALLOF requires that all of the following expressions
   be TRUE for the rest of the instruction to be executed.)
IF :KEY = CHAR 8 THEN SETX XCOR - 1 STOP
   ; moves the turtle 1 step to the left when the left arrow
   is pressed by itself.
IF ALLOF (PADDLEBUTTON 0 ) (:KEY = "S )
   THEN REPEAT 4 [FORWARD 40 RIGHT 90]
   STOP  ;draws a square if Open-Apple S
   is pressed.
IF :KEY = CHAR 27 THEN PRINT [THANKS FOR
   PLAYING.] TOPLEVEL
   ; Esc ends the program.
END
```

Type INSTANT to experiment with these keystrokes. The program only tests for F, R, space bar, Return, Tab, Open-Apple left arrow, left arrow, Open-Apple S, and Esc; all other keys are ignored. You can add other lines, but be aware of the order of the instruction lines, which can be significant. For example, in the above example, the combination of Open-Apple and left arrow must be tested before the left arrow is tested by itself; otherwise the procedure will stop before the Open-Apple and left arrow combination is tested.

Another handy use of ASCII codes is in this procedure, which prompts the user to press a key to go on:

```
TO GO.ON
PRINT1 [Press any key to go on (or ESC to
   quit):]
IF READCHARACTER = CHAR 27 THEN TOPLEVEL
   ELSE CLEARTEXT
END
```

Use these examples as models for creating your own programs that do more interesting things!

Now you're getting to be a Logo expert yourself! Go show your students and colleagues a thing or two!

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 Court
Portland, MA 04103

Implementing Powerful Ideas - The Case of Run

by Rina Zazkis and Uri Leron

Introduction

When teaching computing science—or any other topic with any depth—a fundamental conflict arises. How are we to render our teaching both professionally respectable and educationally sound at the same time? In other words, how are we to accommodate the tastes of both the professional and the student? Unfortunately, the main reasons for the importance of an advanced programming construct—such as aiding abstraction, controlling the complexity of large software systems, or increasing the expressibility of the language—are almost always unappealing and hard to explain to novices.

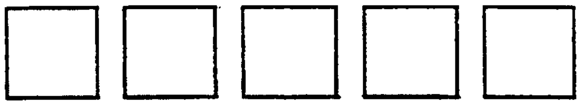
This conflict poses a challenge to the creativity of teachers and curriculum developers: Given a particular non-trivial topic you wish to teach, devise activities for the learner in which both perspectives are genuinely represented. Such 'generic' activities are useful as bridges between the novice's needs and the high—level views of the expert. It is therefore important that the computing education community create as large a pool as possible of such generic activities for the various topics typically appearing in computing courses.

During the years we have been working as teachers, teacher educators, and curriculum developers, our group at the Israeli Logo Centre have collected many such activities, some from the literature, some home-made. In this article we wish to exemplify the above considerations with one classroom idea, which we have found to be particularly effective in its role as bridge. The example is an activity for introducing the RUN construct in Logo (which takes as input a list of Logo commands and executes them), in which the above conflict is particularly pronounced: RUN is considered very powerful by experts, but this power is hard to convey to novices.

Our example can thus be viewed at two different levels. On one level it is simply a classroom idea. On another level, the example represents a bridge—spanning at one end the student's need for a concrete, engaging activity, but at the other end—showing how a particular programming construct can enhance the expressive power of the language. In other words, our example, while simple and engaging to the student, is yet sophisticated enough to require enriching the language to be able to capture the new pattern.

A classroom idea: Variations on a theme

What follows is a description of a classroom activity for introducing RUN, with some educational and computational observations added to it. The general direction of the activity is from a very free, "creative" stage, gradually to a more focused and purposeful discussion.



We first present to the students the picture shown above — five squares in a row—and the procedure generating it. (It is also possible to let them program it, but for experienced students this is a well-known activity, and we prefer to concentrate on the task at hand.)

```
TO ROW.OF.SQUARES
MOVE.TO.START
REPEAT 5 [SQUARE MOVE]
END

TO SQUARE
REPEAT 4 [FORWARD 30 RIGHT 90]
END

TO MOVE.TO.START
CS
LEFT 90
PU
FORWARD 100
PD
RIGHT 90
END

TO MOVE
RIGHT 90
PU
FORWARD 40
PD
LEFT 90
END
```

We then ask them to invent variations on the theme of the picture.

This is done in two stages, each consisting of a short classroom discussion, then some lab work in small groups, then again a "plenary" discussion in which the work of the small groups is shared. In the first stage, the students are encouraged to generate as many—and as wild—variations as they can think of, with no attention paid to programming issues. In the second stage we go over the list collected in the first one, and try to see which of the suggested variations can be realized by adding an input variable to the above procedure. For brevity, we present here the results of the two rounds together.

Typically, students come up with suggestions such as the following:

- Vary the size of the squares (introduce a :SIZE variable). (In the following we shall further abbreviate these two stages as: "Vary the :SIZE of the squares".)
- Vary the distance between adjacent squares.
- Vary the color of the squares.
- Vary the starting :POSITION of the drawing.
- Vary the :SHAPE of the repeating figure. (Note: This seemingly innocent idea is what we are after, but our skilled teacher menages to keep a straight face so as not to lose the creative momentum...)
- Vary the :NUMBER of the squares, etc.

Next, students try to implement the above ideas (or some of them) at the computer. We note that except for the case of the :SHAPE variable, none of the variations poses any serious problem to students with reasonable knowledge of procedures and variables. Following the actual classroom practice, but in a much abridged and streamlined fashion, we now present an analysis of the remaining case of :SHAPE, discussing first our goal, then the difficulties it poses, and, finally, the new means (the RUN instruction) for achieving it.

We wish to have a procedure that will accept as input a shape (a triangle, a square, a house, a mouse, etc.) and draw a row of such shapes. Starting from the given and successfully running ROW.OF.SQUARES, probably the most natural attempt is to replace the SQUARE in it with a variable :SHAPE. We are thus led to consider the following procedure:

```
TO ROW.OF.SHAPES :SHAPE
MOVE.TO.START
REPEAT 5 [:SHAPE MOVE]
END
```

Having written this procedure, a problem immediately presents itself. In what form shall we execute it? (We assume that the shapes in question are already programmed as procedures in Logo.) Again, we follow what's natural. Here are the three ways such a procedure could possibly be executed, together with the resulting Logo error messages:

ROW.OF.SHAPES SQUARE
•(a square is drawn and ...) SQUARE didn't output to ROW.OF.SHAPES
ROW.OF.SHAPES "SQUARE
•I don't know what to do with "SQUARE
ROW.OF.SHAPES [SQUARE]
•I don't know what to do with [SQUARE]

We digress for a brief explanation (in anthropomorphic terms) of these error messages. In the first case, ROW.OF.SHAPES knows it needs an input. Not getting it directly, it hopes to get it as an output from the expression that follows. It therefore executes SQUARE (which accounts for the drawing) and then, when SQUARE fails to output, it issues an appropriate complaint. The two other cases are similar to each other, and we shall explain only the third. Here the input is the list [SQUARE], which becomes the value of the SHAPE variable. The resulting error message "I don't know what to do with [SQUARE]," literally explains what happens when, in the course of the execution, Logo meets the object :SHAPE.

Indeed, what do we want Logo to "do with [SQUARE]"? Why, RUN it, of course! And in Logo we say just this:

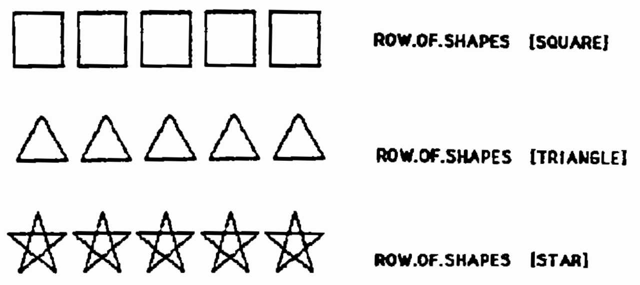
```
RUN [SQUARE]
```

Or, using our variable:

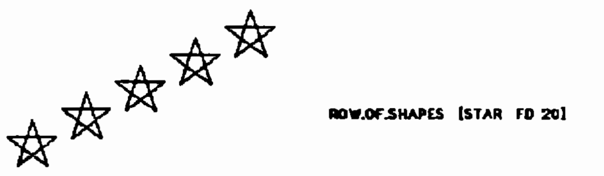
```
RUN :SHAPE .
```

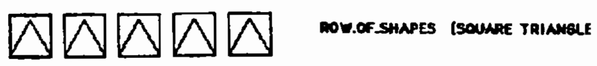
Here is the complete procedure to produce the illustration below:

```
TO ROW.OF.SHAPES :SHAPE
MOVE.TO.START
REPEAT 5 [RUN :SHAPE MOVE]
END
```



In general, RUN takes one input, which must be a list of Logo expressions, and executes them as if they were typed from the keyboard. Below are some more examples.





Looking back on the above classroom activity, we note several features that deserve further mention. To discuss these features it is convenient to group them under the separate headings of educational and computational concerns. To us, however, the most interesting feature of this approach is a mixture of the two, i.e., the combination of a soft, activity based, entry to teaching RUN, with a genuine demonstration of its power.

The computer science perspective

Procedures as data.
The main power of RUN as a programming construct is that it enables us to use procedures as data, making it possible to pass them as inputs and outputs to other procedures. In fact, what we achieve here is blurring the classical distinction between "passive" data and "active" procedures. This powerful capability of the language is automatic in LISP, and in fact, is considered one of the main sources of its power (Abelson and Sussmann, 1985). To recapture it in Logo, as demonstrated above, one first masks the relevant procedure as data by bracketing it, only to be later unmasked with RUN when it is to be executed.

Deferred execution.
Another way of looking at the above maneuver, is as a method to prevent the procedure from being executed prematurely (cf. the unsuccessful attempt ROW.OF.SHAPES SQUARE above). This method is sometimes referred to as "deferred execution." According to this view, the purpose of bracketing the procedure's name is to prevent the Logo evaluator from executing it when it is first met. We can thus perform various operations on it and finally have it executed by applying RUN at precisely the right moment.

Expressive power.
Perhaps the most important aspect of the use of RUN is that it enhances the expressive power of the language, enabling us to capture in it patterns that were previously inaccessible. Without the benefit of RUN we can capture in the language individual patterns like "row of squares" or "row of stars," but the more abstract pattern "row of similar shapes" is beyond our descriptive power. The same phenomenon, incidentally, is encountered much earlier when introducing input variables: We can describe in Logo individual squares (of length 50, 100, etc.), but without variables, we cannot express the abstract idea of "any square." Since input variables considerably enhance the expressive power of the language, a major way to further extend that power is by extending the concept of variable (via RUN) to encompass procedures as well.

The educational perspective

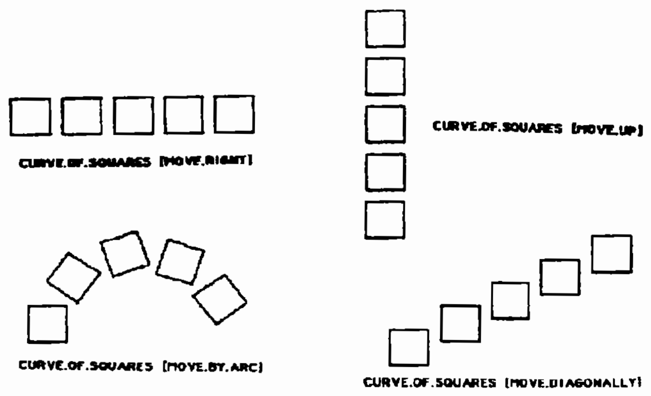
Consider the following by-now-classical approach to introducing input variables to beginners. The students draw squares of various sizes, writing a separate procedure for each. On reflection, they discover that their procedures actually all look the same, except the input to FORWARD. It is then quite natural to introduce a variable to stand for "any desired number of turtle-steps."

Our approach to introducing RUN is continuous with this earlier approach. Here again we study "variations on a theme," this time varying the shape in the repeating pattern. Having written the separate procedures ROW.OF.SQUARES, ROW.OF.STARS, ROW.OF.MICE, etc., the similarity between them invites replacement of the individual shapes with a :SHAPE variable.

On a more general level, this activity also demonstrates the technique of creating a need for a new idea, even temporary frustration, before dumping it on the helpless students.

How is RUN introduced by other sources? All authors seem to agree that the mere definition ("RUN takes one input, which must be a list of Logo expressions, and executes them as if they were typed from the keyboard") is not a very useful introduction. Thus they proceed to supplement it with examples that aim to show its power (e.g., INSTANT in Abelson, 1982, p.152; CALCULATOR and WHILE in Apple Logo II *Reference Manual*, p.138). All these examples, while genuine demonstration of the power of RUN, are not given to the kind of variations-on-a-theme activity we have found to be so effective. They also mostly belong to advanced programming topics that are not very appealing to the average beginner.

We conclude with an additional variation on our original theme that has proved to be an interesting and useful follow-


CURVE.OF.SQUARES [MOVE.RIGHT]
CURVE.OF.SQUARES [MOVE.UP]
CURVE.OF.SQUARES [MOVE.BY.ARC]
CURVE.OF.SQUARES [MOVE.DIAGONALLY]

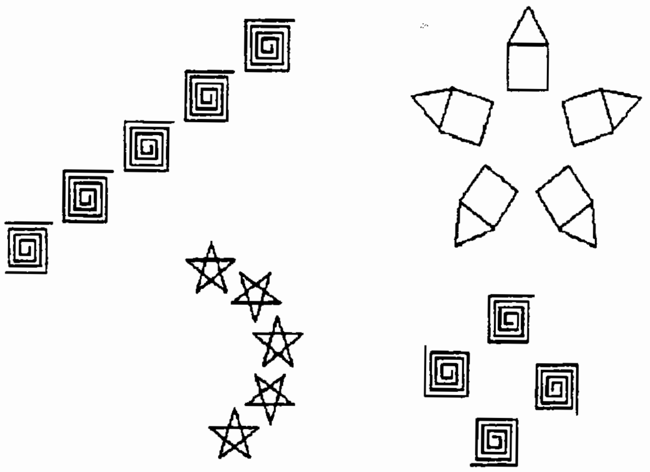
up activity for the students. This variation often actually appears in the first round of students' suggestions, but carrying it out is best postponed to a later stage. The idea is to vary the curve on which the repeating squares are located.

Here is the procedure:

```
TO  CURVE.OF.SQUARES  :MOVE
MOVE.TO.START
REPEAT 5 [SQUARE  RUN :MOVE]
END
```

Finally, combining the shape and the curve variations we get the procedure CURVE.OF.SHAPES :SHAPE :MOVE .

The illustration below gives a few examples of the great variety of patterns that this single two-line procedure can express. (Note the four rectangular spirals in the last drawing. Can you explain what happened to the fifth one?)



References

Abelson, H. (1982). *Logo for the Apple II*. New York: BYTE/ McGraw Hill.

Abelson, H. and Sussman, G. J. (1985). *Structure and interpretation of computer programs*. Cambridge, MA: MIT Press.

Rina Zazkis and Uri Leron
Department of Science Education
Technion - Israel Institute of Technology
Haifa 32 000, Israel
Bitnet: ttr0128@technion

Logo LinX

Reach Out and Touch Logo
by Judi Harris

Does paddling have a place in the Logo classroom? Certainly...when it is the computer's paddles that are being used. Paddle input is supported by most versions of Logo, and can be anything but painful.

Joysticks, game paddles, and touch-sensitive graphics tablets (such as the KoalaPad) are often used as alternative input devices for game and graphics programs. All are connected through a microcomputer's game port or game adaptor, and can be directly accessed with two Logo primitives.

With these inexpensive peripherals, pre-readers, physically challenged students, and those of us that just like to "piddle-paddle" can use Logo in a host of unique and exciting ways. Command of just two Logo commands may eliminate the need to purchase pre-programmed software that accesses touch tablets, and cannot be easily tailored to meet individual student needs.

In this article, I would like to concentrate on Logo interfaces and applications with graphics pads such as the KoalaPad, Animation Station, and Touch Window.

Touching Primitives

When someone touches a graphics pad connected to a microcomputer, two types of information can be detected and acted upon. The two-dimensional position of their finger on the surface of the tablet can be registered with the PADDLE command. The BUTTON? or BUTTONP (in Terrapin Logo, PADDLEBUTTON) command can also be used to determine if graphics pad buttons are being pressed.

Paddle information is typically represented by numbers ranging between 0 and 255. PADDLE 0 outputs position information along the X (horizontal) axis; PADDLE 1 numbers refer to Y (vertical) axis position. A simple recursive procedure can be used to print paddle information on the screen as you move your finger or a stylus over the surface of a graphics pad:

```
TO PADDLE.POS
CT
PRINT SENTENCE [PADDLE 0:] PADDLE 0
PRINT SENTENCE [PADDLE 1:] PADDLE 1
PADDLE.POS
END
```

Button information is output as either "TRUE or "FALSE; the former if the touch pad button indicated is being depressed, the latter if it is not.

```
TO BUTTON.PRESS?
CT
PRINT SENTENCE [BUTTON 0:] BUTTON? 0
PRINT SENTENCE [BUTTON 1:] BUTTON? 1
BUTTON.PRESS?
END
```

Paddle information is most commonly used to sense position of contact with the graphics tablet. Button information is typically used in a conditional statement that allows a user to select an option (such as a screen change or sound effect) whenever s/he chooses.

A Teacher's Touch

If position on the surface of the graphics tablet can be detected with Logo commands, why not correlate the position of the turtle on the screen with the location of the finger or stylus on the touch-sensitive pad?

At first glance, this seems simple enough:

```
SETPOS SENTENCE ( PADDLE 0 )( PADDLE 1 )
```

or, in Terrapin Logo

```
SETXY ( PADDLE 0 )( PADDLE 1 )
```

OOPS! PADDLE 0 inputs range from 0 to 255, but X-axis screen coordinates span approximately -140 to 140 or -120 to 120, depending on the version of Logo that is being used. If SETPOS is used with non-adjusted PADDLE 0 numbers, the turtle could only assume X axis positions between 0 and 255. Negative coordinate placements would be omitted, and screen boundaries would would be ignored. There is a similar discrepancy with PADDLE 1 and Y-axis numbers.

Simple tool procedures that recalculate the range of paddle information and offset the turtle's screen position relative to sizes of different graphics pads can be used to correct the discrepancies.

For the KoalaPad:

```
TO X.POINT
OUTPUT (( PADDLE 0 ) - 131 ) * 1.078
END

TO Y.POINT
OUTPUT ((PADDLE 1 ) - 130 ) * - 0.975
END
```

For Animation Station:

```
TO X.POINT
OUTPUT (( PADDLE 0 ) - 128 ) * 1.025
END

TO Y.POINT
OUTPUT ((PADDLE 1 ) - 128 ) * - 0.86
END
```

For the Touch Window:

```
TO X.POINT
OUTPUT (( PADDLE 0 ) - 125 ) * 1.14
END

TO Y.POINT
OUTPUT ((PADDLE 1 ) - 120 ) * - .73
END
```

Placing the turtle at newly-calculated positions is a simple matter. Since screen coordinates are usually represented by integers (numbers without decimals), X.POINT and Y.POINT output should be simplified with the INT (integer) command (as in POINT, below) before setting the turtle's position to X.POINT and Y.POINT values with PLACE.TURTLE.

```
TO POINT
OUTPUT LIST ( INT X.POINT )
( INT Y.POINT )
END

TO PLACE.TURTLE
SETPOS POINT
PLACE.TURTLE
END
```

The turtle's screen position will now reflect changing points of contact on the graphics tablet.

Touchy Areas

X.POINT and Y.POINT information can also be used to delineate sensitive areas on the tablet. This is especially appropriate for Touch Window applications, since this type of graphics pad is translucent, and can be mounted on the front of a monitor, allowing users to see what is displayed on the screen through the graphics tablet itself.

Suppose that we wanted to divide the screen/tablet area into four sections, so that a physically impaired child would have to touch each screen section to see a picture displayed inside it. Four procedures could be written as follows:

```
TO SECT1?
OUTPUT AND ( X.POINT < 0 )
( Y.POINT > 10 )
END

TO SECT2?
OUTPUT AND ( X.POINT > 0 )
( Y.POINT > 10 )
END

TO SECT3?
OUTPUT AND ( X.POINT < 0 )
( Y.POINT < 10 )
END

TO SECT4?
OUTPUT AND ( X.POINT > 0 )
( Y.POINT < 10 )
END
```

These procedures could then be used as conditional input.

```
TO TOUCH.PICTURES
IF SECT1? [LOADPIC "UPPER.LEFT]
IF SECT2? [LOADPIC "UPPER.RIGHT]
IF SECT3? [LOADPIC "LOWER.LEFT]
IF SECT4? [LOADPIC "LOWER.RIGHT]
TOUCH.PICTURES
END
```

A Touch of Creativity

An interesting programming challenge might be to write a procedure that would accept four touches as the corners of a sensitive area, then automatically define a SECT?-like procedure. Sue Anderson, teacher of preschool handicapped children in Albemarle County, Virginia, conceived and solved this problem so that she could use a Touch Window in conjunction with a Logo-controlled videodisc player. Now when the speech synthesizer (also driven by Logo) tells her students to "touch the gorilla's belly," they can look through the clear graphics tablet at the videodisc image and touch the area that she defined with her time-saving tool procedure.

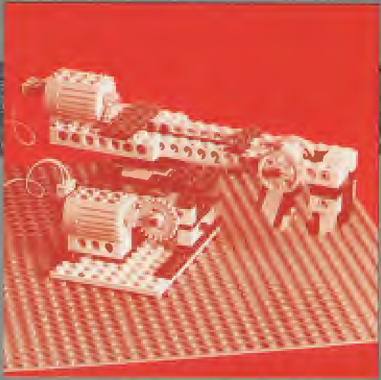
Other results for touching sensitive areas can be programmed easily. For example,

- Different musical notes could play when the appropriate lines or spaces were touched on a screen display of the musical staff.
- Maze walls could buzz when the turtle makes contact with them.
- Printed words (such as "cat," "truck," "blue," and "green")

Gear Up For Learning

Explore

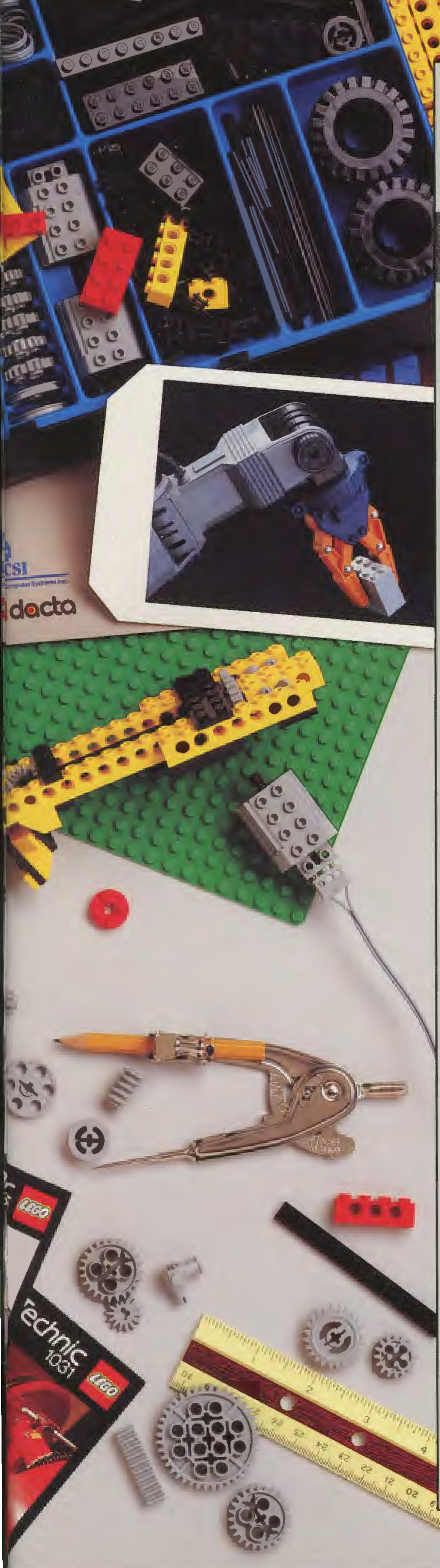
Investigate



Apply

LEGO dacta

1031
Technic
LEGO



Philosophy of Learning

I want children to reinvent the wheel.

Of course, they are going to reinvent it anyway as part of their own learning experience. But they will do so all the more richly if wheels and talk about wheels and interest in wheels are well represented in their learning environment . . . and, above all, if the significant people in their lives sincerely believe in them as discoverers.

A full appreciation of children as inventors does not come easily. In my book, *Mindstorms*, I wrote about how reinventing gears and making my own theories about these particularly wonderful wheels helped me grow up to be a mathematician. But it took me half a lifetime to arrive at that appreciation of my own experience, and the process is by no means over. Every day still brings new insights into my own learning and with it the joyous sense of deeper empathy with other learners.

You can learn a lot about inventing and about learning by observing children. But to go further you have to jump in and do it yourself. What is most wonderful about the LEGO materials is that they allow everyone, young or old, naïve or sophisticated, boy or girl, to use the same stuff as a medium for invention.

Why not pursue your own reinvention of the wheel? Think of all the questions you can ask. Why do some vehicles have four wheels and some three and some two? Or one? What good are wheels anyway? They make the cart easier to pull . . . but why? How could any one person know all the answers?

Or, think about the many different kinds of wheels. In the LEGO materials, there are just plain wheels, gear wheels and pulley wheels. All these make things move, but there is also a black and white sectored wheel for measuring motion rather than making it happen.

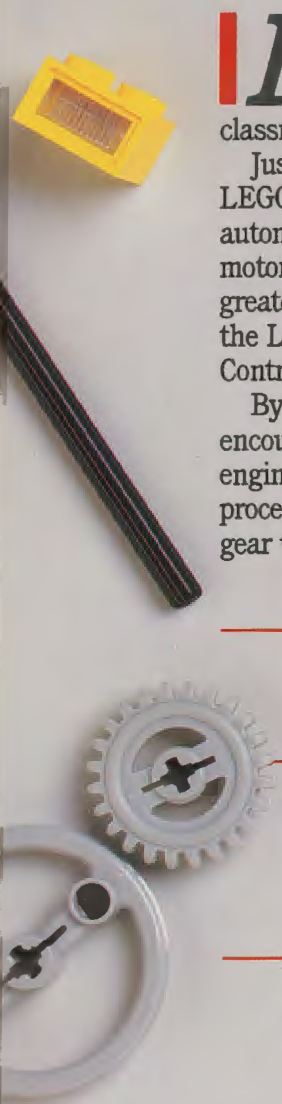
Then think of the wheels you can build: ferris wheels, waterwheels, paddlewheels, steering wheels, flywheels and the kind of activity wheel people put in a hamster's cage. Don't you think you might reinvent some more?

You may discover for yourself that, like a wheel, learning can go on and on. Or, even better, your students might help you make this discovery together!

[illegible]

Seymour Papert
LEGO Professor of Learning Research
Massachusetts Institute of Technology

Introducing LEGO® Dacta



LEGO Dacta, the educational division within the world famous LEGO Group, brings excitement to today's classrooms! LEGO bricks, motors, lights, sensors, pulleys, gears and computer controls are combined into a learning system of innovative classroom sets for students in grades 3 through 12.

Just like shifting gears on an automobile, students can progress through the system of LEGO Dacta educational products. The Technic sets are like the *lower* gears used to get the automobile moving. Technic I introduces simple machines; Technic II adds the concept of motorization. Technic Control sets are like the *higher* gears used to take the automobile to greater speeds. Technic Control 0 introduces computerized control using a special version of the Logo computer language. Technic Control I extends computerization to robotics. Technic Control II provides for the construction and operation of computerized measuring instruments.

By the time students progress through the entire LEGO Dacta system, they will have encountered important concepts from mathematics, physical science, physics, robotics, engineering and artificial intelligence. In addition, they will have participated in important processes such as problem solving and cooperative learning. This is why we say, "Students gear up for learning with LEGO Dacta educational products."

Quality

LEGO products are known world-wide for their high quality. LEGO Dacta sets stand up to many years of classroom use, and carry a lifetime guarantee.

Learning System

LEGO Dacta products are organized into a learning system ranging from Technic sets for simple machines up through Technic Control sets for robotics, physics and artificial intelligence. Moreover, each set is fully compatible with the others, enabling students to build even more complex projects.

Hands-On

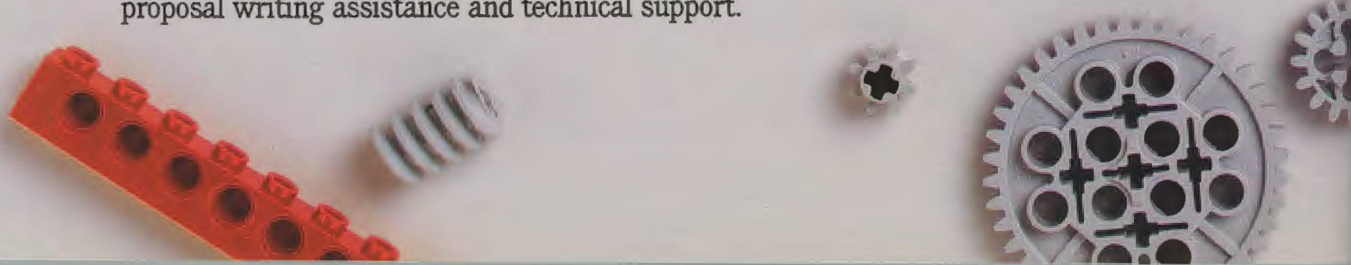
LEGO Dacta products provide students with unlimited opportunities for hands-on learning. They can explore, investigate and apply what they have learned through building with LEGO Dacta sets.

Storage

LEGO Dacta Technic and Technic Control products are packaged in special trays and storage units to simplify classroom management. Several sets have transparent lids, permitting an instant inventory.

Curriculum Support

LEGO Dacta Technic and Technic Control sets are accompanied by student activity cards and other teacher support material. LEGO Dacta educational specialists provide training, teaching ideas, curriculum connections, grant proposal writing assistance and technical support.



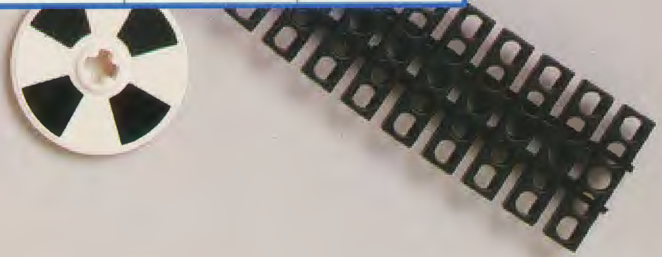
Overview

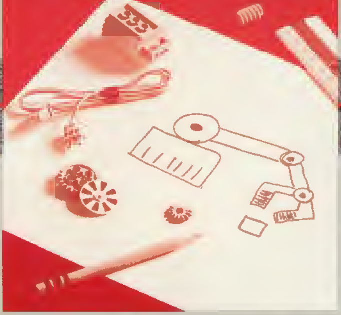
Technic Products	Curriculum Areas	Grade Levels	Page
Technic I Set & Lesson Plans Items 1030, 1031, 1035, 999	Simple Machines Physical Science	3–9	6
Technic I Activity Center Item 9603	Physical Science Problem Solving	3–6	8
Technic II Set & Lesson Plans, Pneumatic Elements Items 1032, 1033, 1036	Motorized Machines Technology	4–9	10
Additional Technic Products Items 9605, 1038, 1039		3–9	12

Technic Control Products	Curriculum Areas	Grade Levels	Page
Technic Control 0, LEGO® TC logo Items 951, 966	Science Mathematics Social Studies Technology	4–12	14
Technic Control I Item 1090	Robotics Artificial Intelligence Engineering Technology	7–12	20
Technic Control II Item 1092	Robotics Scientific Measurement Technology	7–12	21
Spare Parts & Technical Specifications			22

All prices for LEGO Dacta products in this catalog are valid through December 15, 1990.

Send orders to:
LEGO Dacta, 555 Taylor Road, Enfield, CT 06082
1-800-527-8339





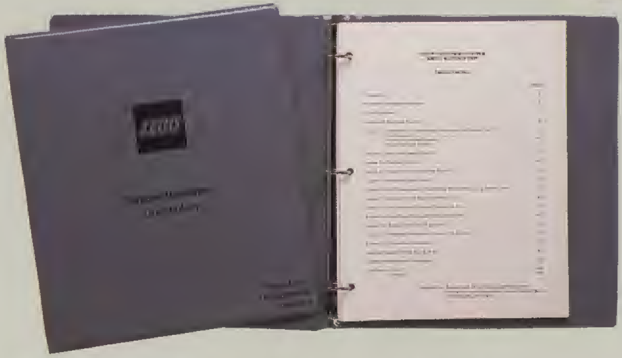
Technic I

LEGO® Dacta Technic I (item 1030) gives students in grades three through nine a hands-on introduction to the fundamental concepts of simple machines, such as levers, gears and pulleys. This award-winning set is one of the most popular of all LEGO Dacta products, and with good reason.

Students explore important concepts by designing and building simple models. They investigate the properties of simple machines by manipulating the models. Practical problems challenge students to apply what they have learned.

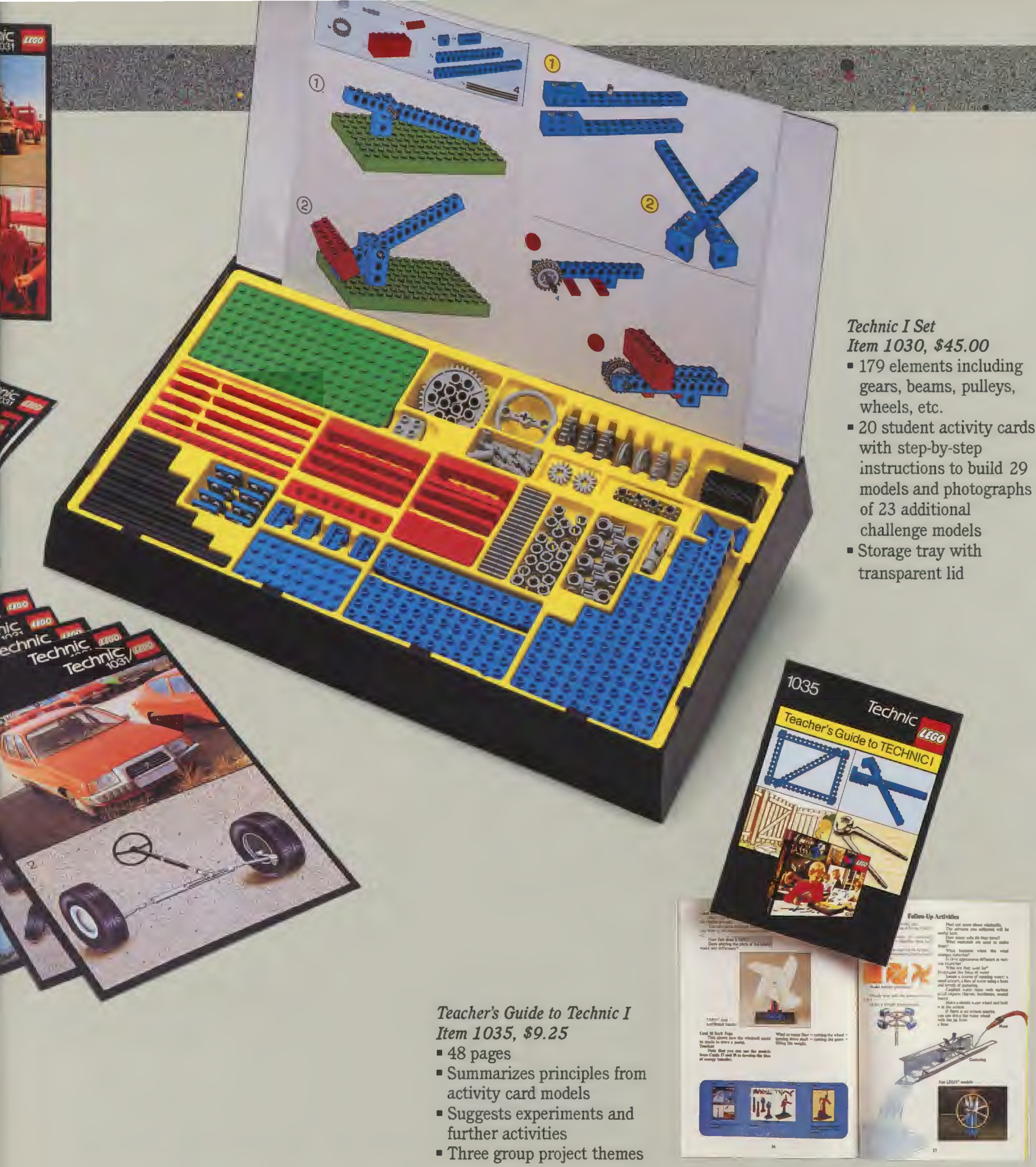
Building LEGO models develops constructional, numerical, graphical and problem solving skills. Group work enhances interaction skills. Active involvement in the successful completion of projects reinforces understanding of concepts and encourages further investigation.

Three-part activity cards supplied with each set feature (1) photographs of simple machines from real life, (2) step-by-step building instructions for simple machine models and (3) photographs of an assembled challenge model based on simple machines. A teacher guide (available separately) contains additional information and teaching suggestions. A fully developed Simple Machines Curriculum is also available.



Simple Machines Curriculum, Item 999, $20.00
- 102 pages
- Developed by a team of teachers from the Anoka-Hennepin County School District of Minnesota
- Includes complete lesson plans for in-depth study of simple machine concepts
- Also includes study masters, review sheets and Team Kit Care Record forms

- *Simple machines: levers, gears, pulleys*
- *Grades three through nine*
- *Designed for two students per set*



Technic I Set
Item 1030, $45.00

- 179 elements including gears, beams, pulleys, wheels, etc.
- 20 student activity cards with step-by-step instructions to build 29 models and photographs of 23 additional challenge models
- Storage tray with transparent lid

Teacher's Guide to Technic I
Item 1035, $9.25

- 48 pages
- Summarizes principles from activity card models
- Suggests experiments and further activities
- Three group project themes

Technic I Activity Center

The LEGO® Technic I Activity Center (item 9603) is designed for classrooms in grades 3 through 6 using the Technic I set (item 1030). The center provides exciting student activities and detailed teacher support for a wide variety of physical science areas.

The Technic I Activity Center is a flexible product, easily organized to suit your teaching needs best. The 110 activity cards are separated by activity type and are extensively cross-referenced for use in six physical science curriculum areas, plus a special enrichment section. Activities are easily reorganized into topic or theme areas, if desired.

Activity types include:

- Explorations
- Investigations
- Applications in Problem Solving:
    - Real World Simulations
    - Inventions

Curriculum areas include:

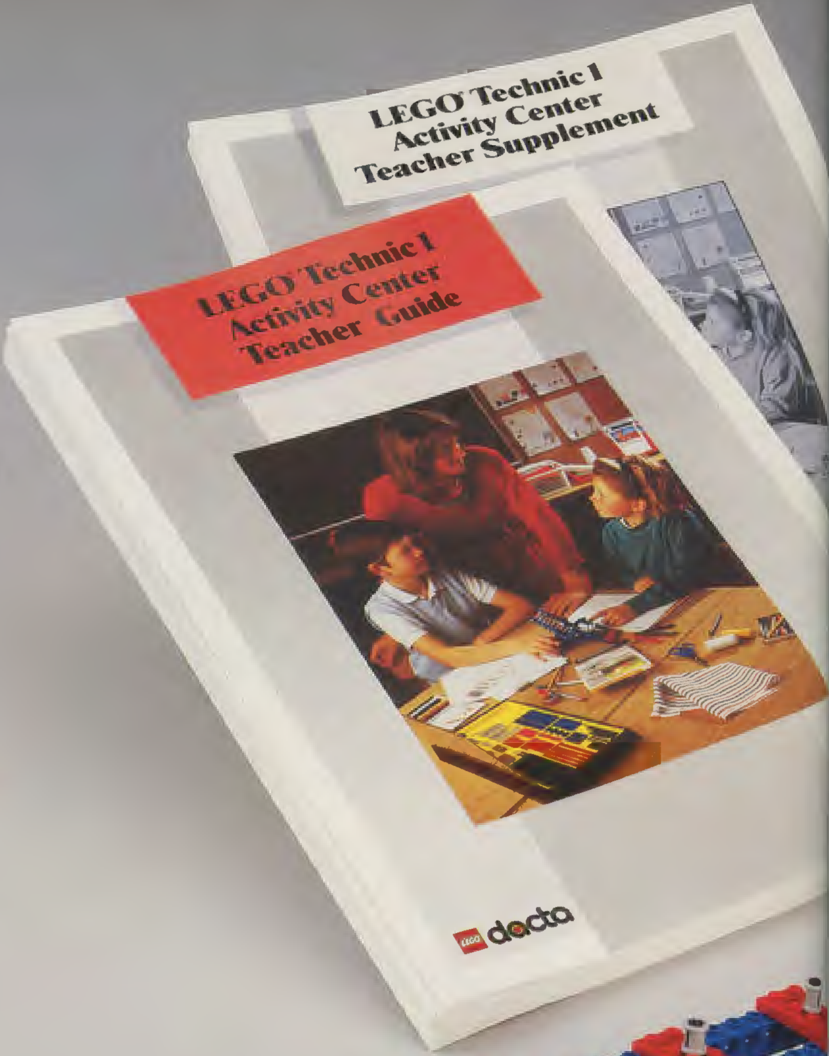
- Forces and Structures
- Levers
- Pulleys
- Gears
- Wheels and Axles
- Energy

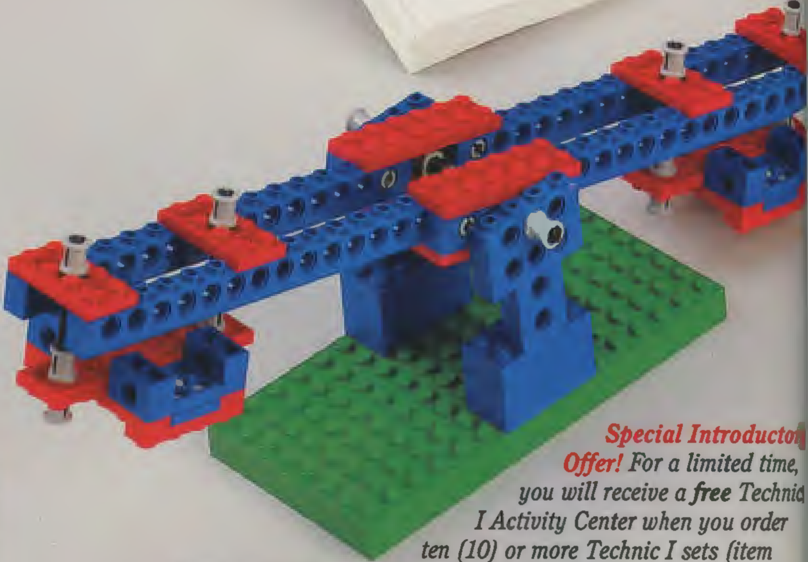
Theme areas include:

- Medieval Castle
- The Farm
- The Harbor
- The Amusement Park
- Getting Around
- The Big Race

The four-color activity cards are accompanied by a comprehensive teacher guide, with setup and organizing suggestions, detailed comments and extension activities for each student card, background information about each of the subject areas, and a number of copy masters for both teacher and student use.

The activity center materials are housed in a sturdy plastic storage container for easy filing and access.



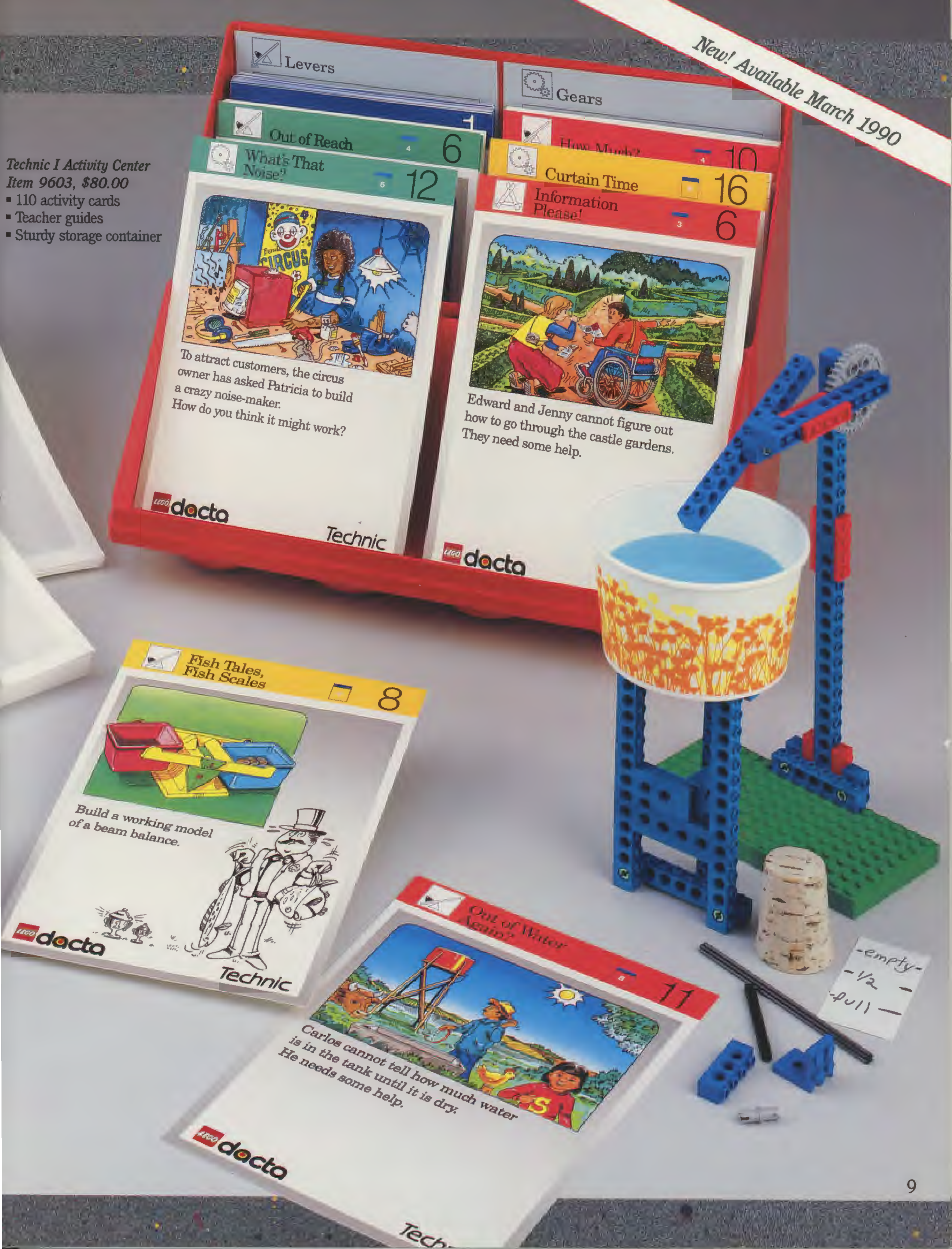


Special Introductory Offer! *For a limited time, you will receive a* ***free*** *Technic I Activity Center when you order ten (10) or more Technic I sets (item 1030). Please include a written request for your* ***free*** *Technic I Activity Center with your order. This offer is valid for orders received between December 15, 1989 and December 15, 1990.*

- *Forces and structures, levers, pulleys, gears, wheels, energy*
- *Grades three through six*
- *Contains sufficient activities for one classroom equipped with Technic I sets (item 1030)*

New! Available March 1990

Technic I Activity Center
Item 9603, $80.00
- 110 activity cards
- Teacher guides
- Sturdy storage container


Levers
Gears
Out of Reach 6
How Much? 10
What's That Noise? 12
Curtain Time 16
Information Please! 6
To attract customers, the circus owner has asked Patricia to build a crazy noise-maker. How do you think it might work?
Edward and Jenny cannot figure out how to go through the castle gardens. They need some help.
Fish Tales, Fish Scales 8
Build a working model of a beam balance.
Out of Water Again? 11
Carlos cannot tell how much water is in the tank until it is dry. He needs some help.
dacta
Technic
-empty-
-1/2 -
-full -

Technic II

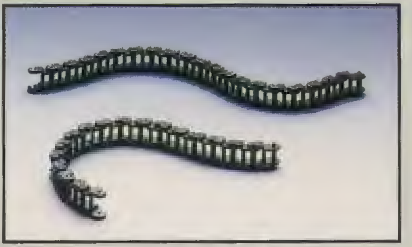
The LEGO® Technic II set (item 1032) is designed to follow the Technic I set in the instructional sequence. It extends the principles of simple machines by adding the concept of motorization. With a 4.5 volt electric motor, students can bring their projects to life.

The Technic II set is excellent for the study of power transmission. It contains a wide variety of power train connectors, such as a chain drive, a differential gear system and a universal joint, in addition to several sizes of regular gears. Supplemental pneumatic elements are also available to add even more versatility to the set.

The Technic II set is growing in popularity with teachers of technical education. It is appropriate for students from upper elementary through the high school grades. The set creates an awareness of the progression of technology and deepens the understanding of mechanical principles through the building of realistic motorized models.

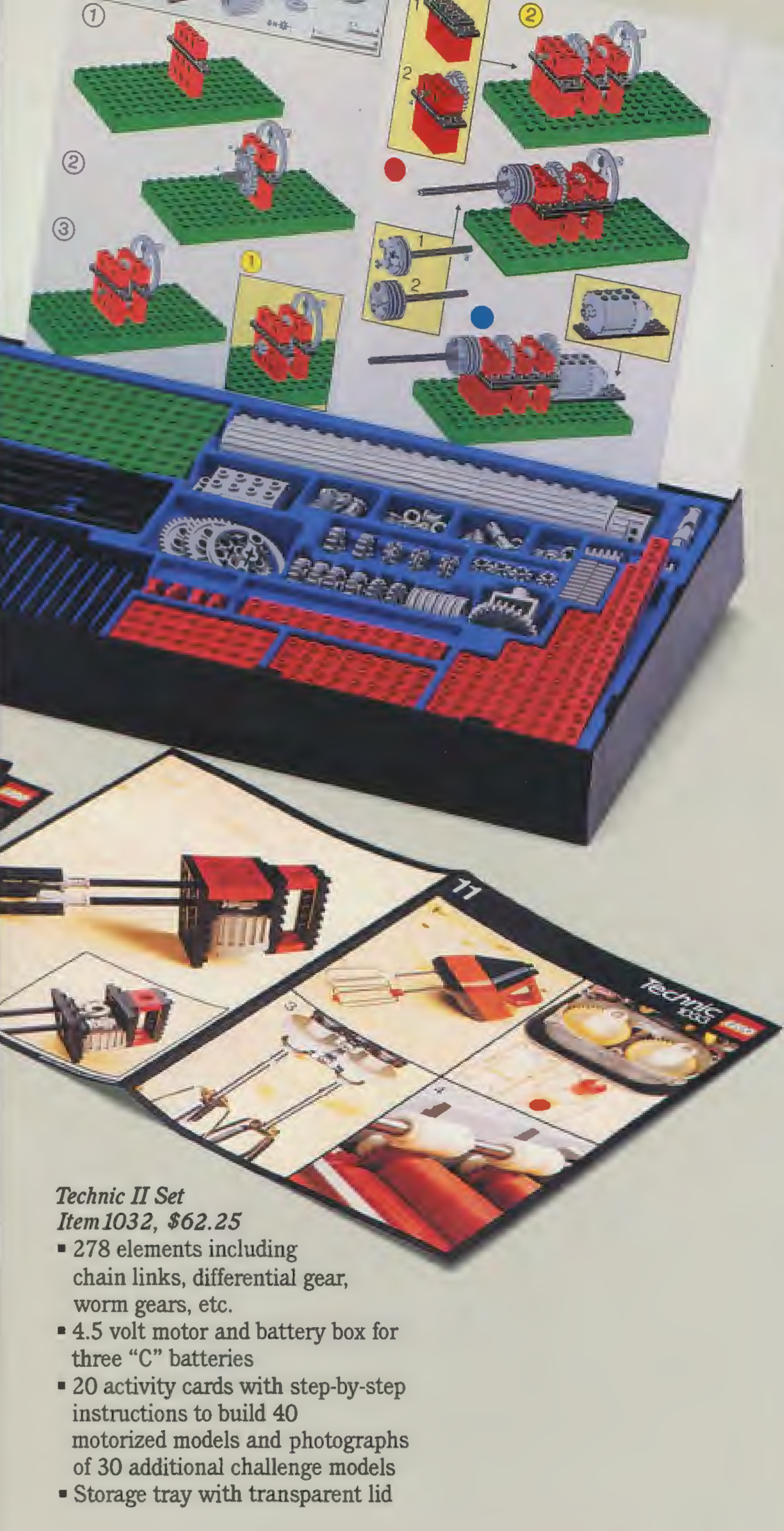
Building motorized models develops students' constructional, numerical, graphical and problem solving skills. Group work enhances interaction skills. Active involvement in motorized projects reinforces an understanding of key concepts and encourages further investigation.

Three-part activity cards supplied with each set feature (1) photographs of motorized machines from real life, (2) step-by-step building instructions for simple motorized models and (3) photographs of assembled challenge motorized models. A teacher guide available separately contains additional information and teaching suggestions.


Chain links included

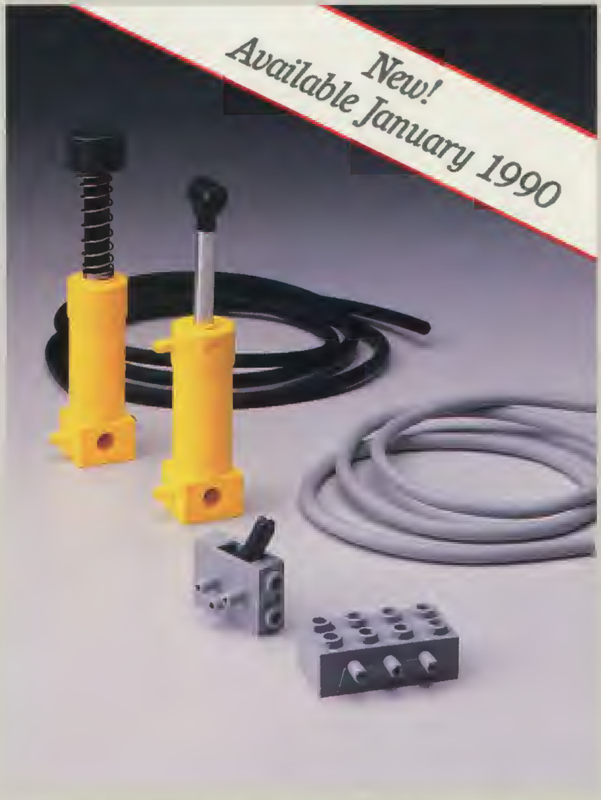
Teacher's Guide to Technic II
Item 1036, $9.25
- 48 pages
- Summarizes principles from activity card models
- Suggests problem solving ideas and further activities
- Three group project themes

- *Motorized machines, power transmission, technology*
- *Grades four through nine*
- *Designed for two students per set*

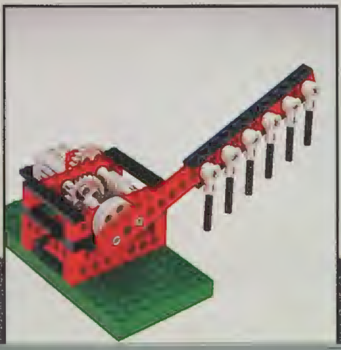


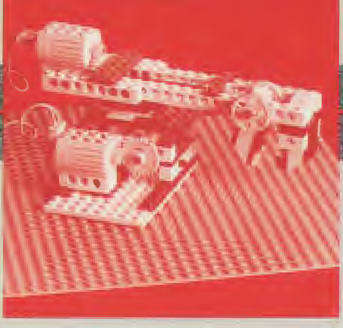
Technic II Set
Item 1032, $62.25

- 278 elements including chain links, differential gear, worm gears, etc.
- 4.5 volt motor and battery box for three "C" batteries
- 20 activity cards with step-by-step instructions to build 40 motorized models and photographs of 30 additional challenge models
- Storage tray with transparent lid



Pneumatic elements are available for use with the Technic II set (item 1032) or other LEGO Dacta sets to introduce another method of controlling moveable parts. Elements include a pneumatic hand pump, two-way piston, valve, switch and assorted hoses. They are compatible with LEGO building elements and can be incorporated easily into LEGO Dacta projects. They are appropriate for use in technical education classes and provide an excellent introduction to the principles of pneumatics. For prices and ordering information, please see page 23.





Technic Supplemental Sets

Several additional Technic items are available to supplement instruction with Technic I and Technic II sets. These items could be provided to students who wish to build especially complex projects or who want to extend their knowledge in new directions.

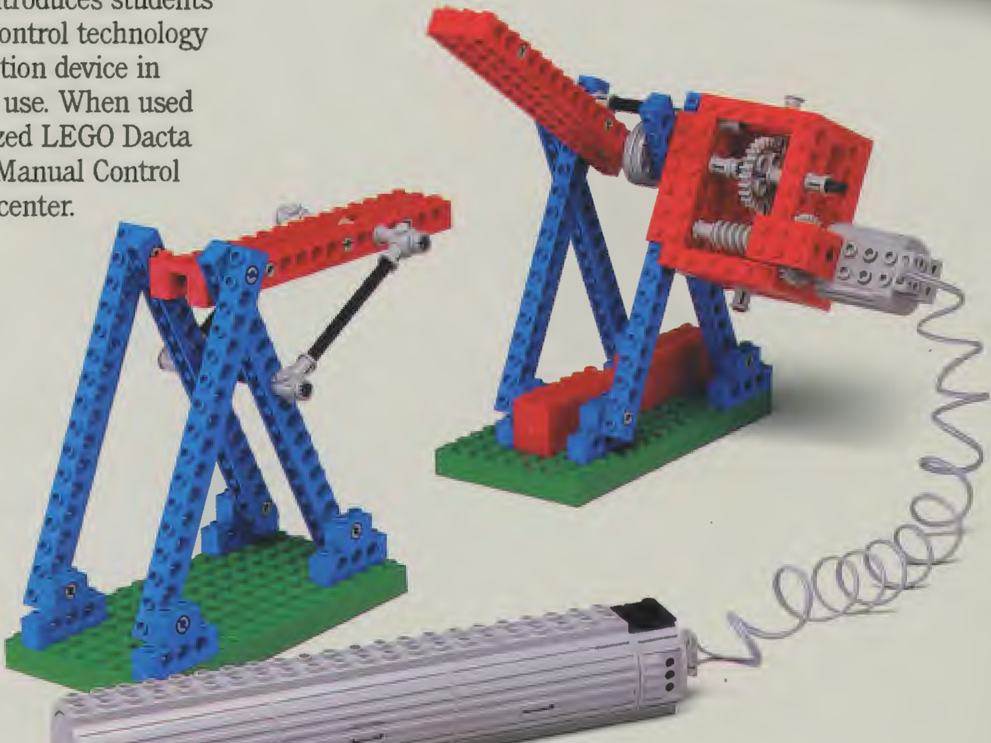
For example, a group of seventh grade students studying physical science may want to design and build a larger project requiring more elements than those available in their Technic II set. In that case, the LEGO® Technic Resource Set (item 9605) could provide the additional elements needed.

Or, perhaps a group of eighth grade technology students want to build and control a steerable motorized transport unit. The Technic Universal Buggy (item 1038) could provide the chassis, and the Technic I set could be used for the superstructure.

For controlling more sophisticated motorized projects, the Technic Manual Control (item 1039) is appropriate. The control panel introduces students to the fundamental concepts of control technology and serves as an excellent transition device in preparing students for computer use. When used in conjunction with a computerized LEGO Dacta set (see next page), the Technic Manual Control provides a supplemental control center.

LEGO Technic Resource Set
Item 9605, $160.00

- 1516 elements, the approximate equivalent of two Technic I (item 1030) and Technic II (item 1032) sets plus about 600 additional elements
- Large storage trays



Technic Universal Buggy, Item 1038, $50.00

- 117 elements including two 4.5 volt motors
- Step-by-step instructions to build two models of a steerable chassis
- Connects to Technic Manual Control or to Technic Control computer interface box

Technic Manual Control, Item 1039, $50.00

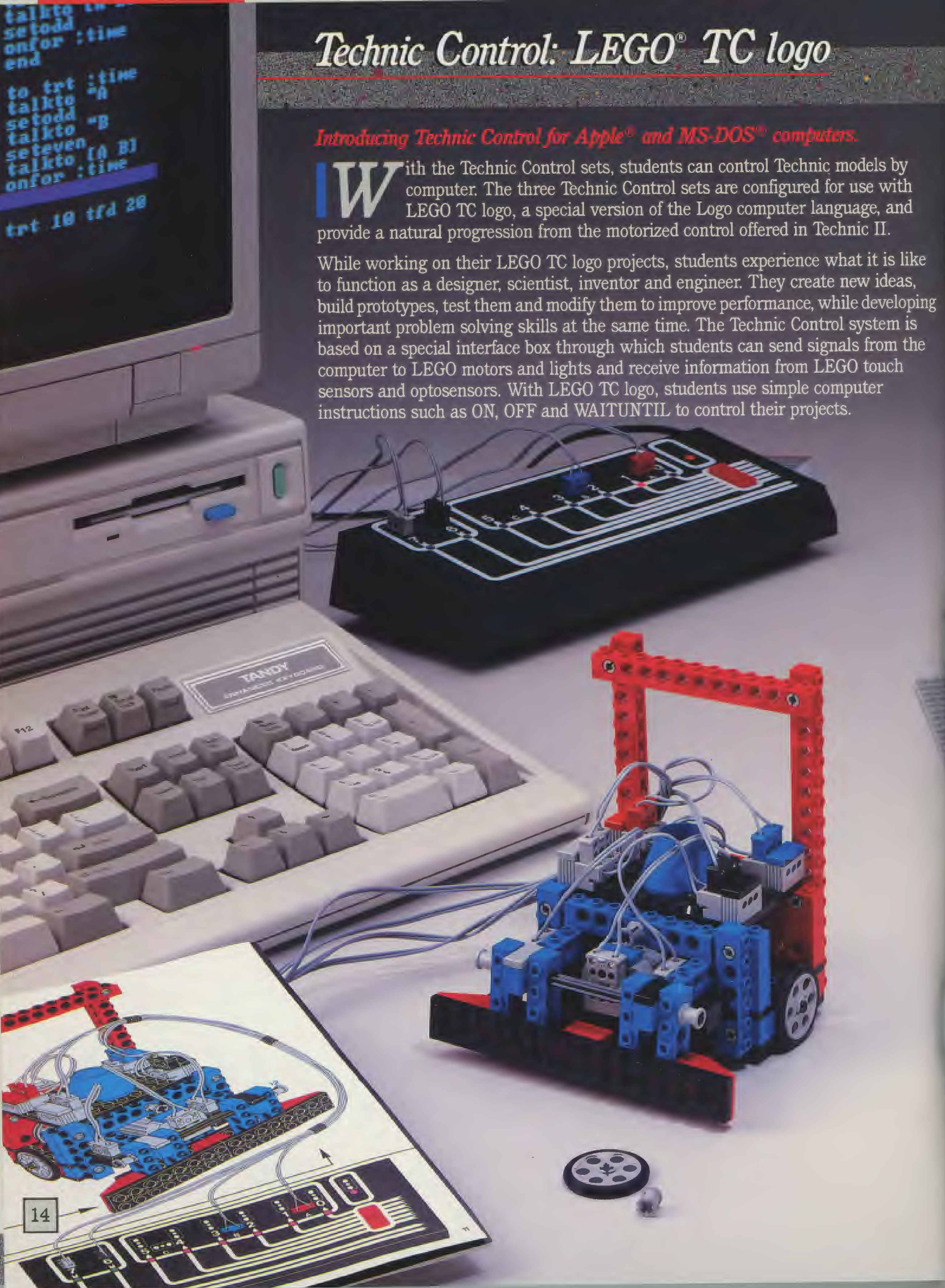
- 39 elements, including three switch panels
- Battery box for three "C" batteries
- Controls up to three 4.5 volt motors
- Step-by-step instructions for assembling different control panels

Technic Control: LEGO® TC logo

Introducing Technic Control for Apple® and MS-DOS® computers.

With the Technic Control sets, students can control Technic models by computer. The three Technic Control sets are configured for use with LEGO TC logo, a special version of the Logo computer language, and provide a natural progression from the motorized control offered in Technic II.

While working on their LEGO TC logo projects, students experience what it is like to function as a designer, scientist, inventor and engineer. They create new ideas, build prototypes, test them and modify them to improve performance, while developing important problem solving skills at the same time. The Technic Control system is based on a special interface box through which students can send signals from the computer to LEGO motors and lights and receive information from LEGO touch sensors and optosensors. With LEGO TC logo, students use simple computer instructions such as ON, OFF and WAITUNTIL to control their projects.



```
talkto
setodd
onfor :time
end

to trt :time
talkto "A
setodd
talkto "B
seteven
talkto [A B]
onfor :time

trt 10 tfd 20
```

LEGO and Logo are alike in many ways. Each is a flexible construction set that gives students freedom to explore, to investigate and to apply what they learn through building. In one case, the building blocks are LEGO bricks; in the other, the building blocks are Logo procedures. In each case, the building blocks fit together in simple ways. From combinations of these simple connections, complex structures can emerge.

As powerful as LEGO and Logo are individually, they become much more powerful when joined together. When students write Logo programs to control their LEGO constructions, they come in contact with important ideas such as sequencing and feedback. The two systems reinforce one another; the powerful elements of LEGO and Logo create one of the richest environments yet for students to explore. The possibilities are as unlimited as a student's imagination.

LEGO® TC logo Starter Pack

The LEGO® TC logo Starter Pack contains all the necessary components for setting up a computerized LEGO building station in the classroom.
Item 951 (Apple) $485.00 / Item 966 (MS-DOS) $515.00

What Do I Need To Get Started?

- *Situation:* One computer for every 2 to 4 students
  Suggested Order: One LEGO TC logo Starter Pack per computer
- *Situation:* One computer for every 5 or more students
  Suggested Order: One LEGO TC logo Starter Pack per computer (2 to 4 students each) plus additional Technic Control 0 sets for remaining student groups

The Technic Manual Control (item 1039) and the Technic Resource Set (item 9605) are excellent supplements to the Starter Pack. (See page 13.)

See page 19 for information on ordering Starter Pack components separately.

Components include:

- Setup guide
- Technic Control 0 building set
- Teacher and student guides
- Software disks and reference guides
- Computer slot card
- Interface box.

(Computer not included.)

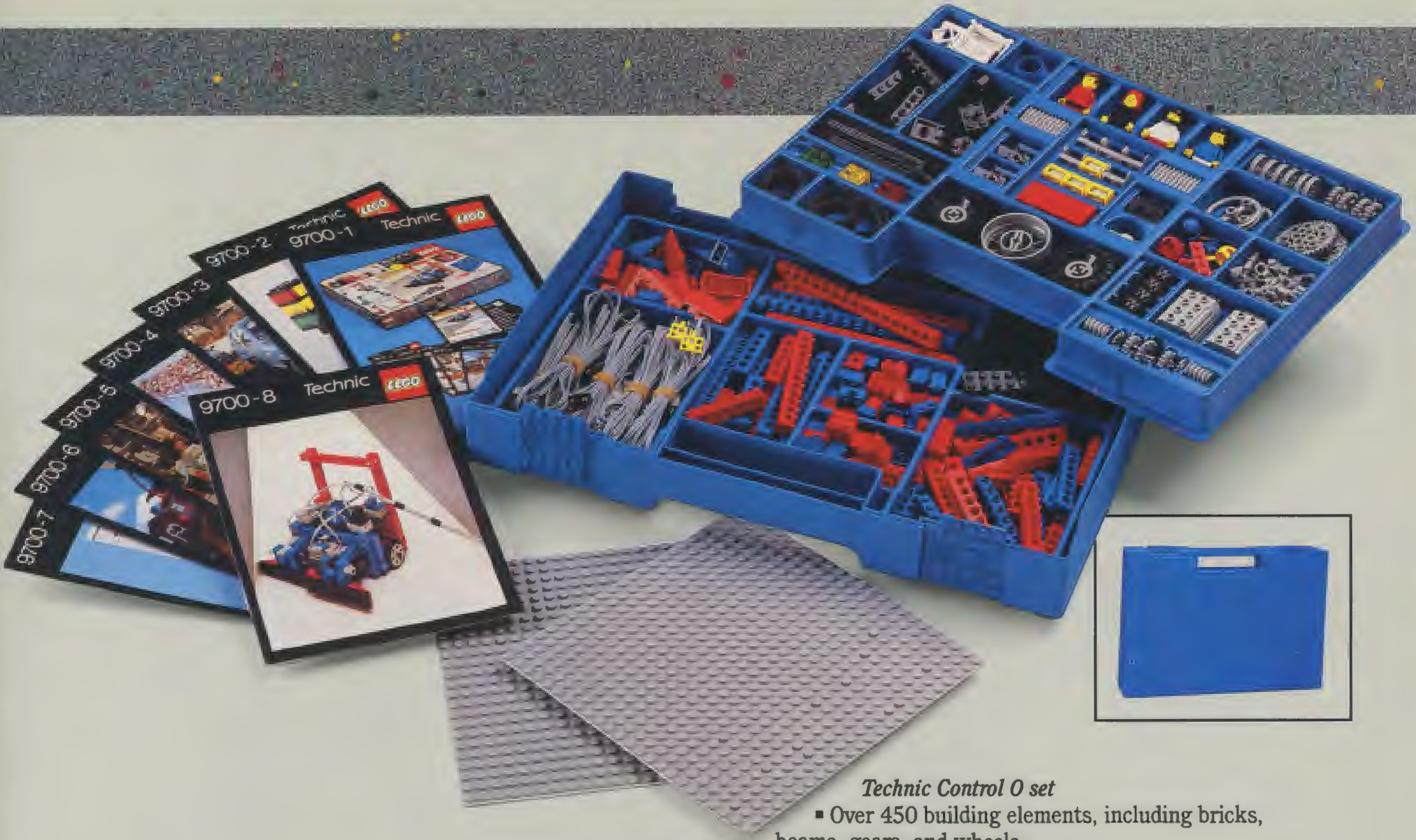
- *Computer controlled models: traffic light, car, starting gate, conveyor belt, carousel, washing machine, robot turtle*
- *Science, mathematics, social studies, technology*
- *Grades four through twelve*
- *Designed for two to four students per set*

Teacher's Guide
- 153 spiral bound pages
- Curriculum correlations for science, mathematics, social studies and computer activities
- Teacher tutorial
- Classroom management suggestions, lesson plans and activities

Reference Guides
- The LEGO TC logo *Reference Guide* is a 134-page manual giving complete explanations and examples of all LEGO TC logo comands.
- The LEGO TC logo *Quick Reference Guide* summarizes the most frequently used commands in a handy format.

Setup Guide
- 13 cards with step-by-step instructions on setting up the components of the Starter Pack to operate with a computer

Student Guides
- Two copies each of three student guides
- *Getting Started*, a 57-page beginner's guide
- *Making Machines*, a 30-page intermediate guide
- *Teaching the Turtle*, a 32-page more advanced guide



Technic Control O set

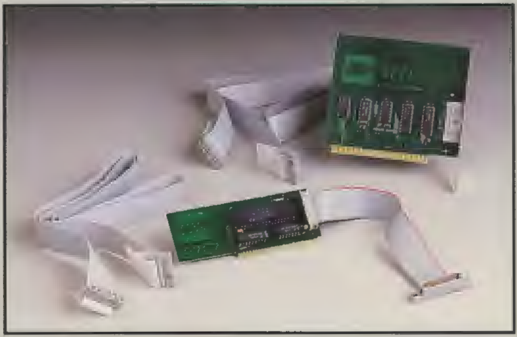
- Over 450 building elements, including bricks, beams, gears, and wheels
- Computer control elements, including two touch sensors, one optosensor, one counting wheel, two motors, and four lights
- 8 student activity cards with step-by-step instructions to build 8 computer controlled models and photographs of 6 additional challenge models
- Carrying case with two storage trays

Software

- Two copyable disks containing the LEGO TC logo computer language, including four screen turtles, simple word processing and printer interface capability.
- Apple: One write-protected master disk, one operating disk, both in 5¹/₄″ format, *or*
- MS-DOS: One 5¹/₄″ master disk, one 3¹/₂″ master disk

Interface Box and Transformer

- The interface box includes six output ports, two input ports and one test port, and independently controls up to three motors or six light bricks and two sensors.
- The transformer provides electrical power to the interface box and plugs into standard electrical sockets.



Slot Card

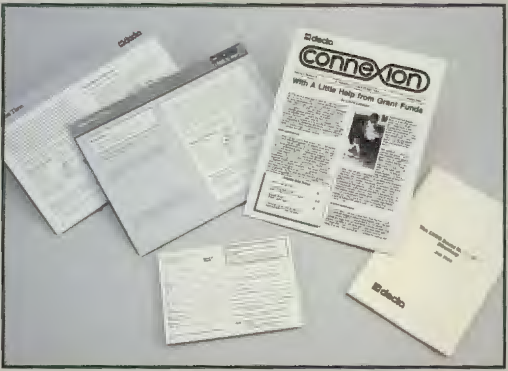
- Apple: Slot card and cable for Apple IIe or Apple IIgs, or other Apple compatible computer, *or*
- MS-DOS: Slot card and cable for IBM PC, IBM PS/2, Tandy® 1000, or other MS-DOS compatible computer

Curriculum Support

LEGO® TC logo connects with curriculum areas such as:

Typical Activity	Science Concepts	Mathematics Concepts	Social Studies Ideas
Motorized Car	Forces Motors, Electricity Gears Mechanical advantage Programming	Time-rate-distance	History of machines Transportation History of the computer Uses of computers
Traffic Light	Light Color Animal senses Machine sensors	Codes Binary arithmetic Recursion	Communication
Merry-Go-Round	Circular motion Centrifugal force	Circles Fractions	History of entertainment
Washing Machine	Time/clocks Using sensors for safety	Logic Time Codes	Automated house House of the future History of household chores Product safety
Robot Turtle	Motion Time-rate-distance Robots Simulation	Measurement Angles Ratios Calibration	Machines vs. animals Animal behavior

LEGO® TC logo users . . . Join The LEGO DACTA CONNEXION™ Free



The LEGO DACTA CONNEXION is the user group for LEGO TC logo. To join, just fill out the registration card in your LEGO TC logo Starter Pack and mail it in. Three times each school year, a special mailing is sent to all members. Each mailing includes exciting new LEGO TC logo activities for students with corresponding teacher guide pages, and the current issue of The LEGO DACTA CONNEXION newsletter. Each issue contains curriculum ideas, feature articles on significant LEGO TC logo projects and events, hints and tips for users, local and global news about innovative projects, and a calendar of LEGO TC logo related conference presentations, workshops and courses. Special coupons are often included for members only, offering extra savings on LEGO Dacta purchases. To promote communication among members, a directory of the names and addresses of LEGO DACTA CONNEXION members is included in the fall mailing.

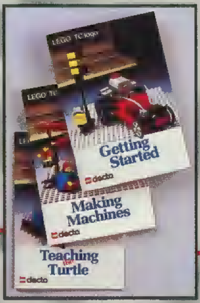
LEGO® TC logo Starter Pack Components

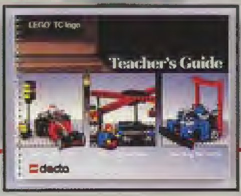
Combinations of the LEGO TC logo Starter Pack components are available for separate purchase as shown below.

Technic Control 0 Set
Item 9700 $142.00
Contains over 450 LEGO elements, including bricks, wheels, pulleys, gears, motors and sensors, enough for one building station, appropriate for 2 to 4 students. Also includes eight sets of building instructions for LEGO TC logo projects.

Slot Card Pack
Item 955 [Apple] $145.00
Item 965 [MS-DOS] $145.00
Contains computer slot card, cable, LEGO TC logo software, *Reference Guide* and *Quick Reference Guide.* To establish a LEGO TC logo computer station, you will also need the Interface Box & Transformer [item 9750 below].

Reference Guide Pack
Item 957 [Apple] $15.00
Item 967 [MS-DOS] $15.00
Contains one copy of the LEGO TC logo *Reference Guide* and the *Quick Reference Guide.*





Interface Box & Transformer
Item 9750 $170.00
Contains the interface box, with input, output and test ports, and a transformer to provide electrical power. The cable from the computer slot card connects to one end of the interface box. Motors, lights and sensors connect to the various ports.

LEGO TC logo Software and Literature
Item 952 [Apple] $110.00
Item 962 [MS-DOS] $115.00
Contains LEGO TC logo software, *Reference Guide, Quick Reference Guide, Teacher's Guide* and two copies of the three student guides. Sturdy storage box included.

Teacher's Guide
Item 953 [Apple] $12.50
Item 963 [MS-DOS] $20.00
One copy of the 153-page LEGO TC logo *Teacher's Guide.*

Student Guides
Item 954 [Apple] $22.25
Item 964 [MS-DOS] $22.25
Contains two copies of three LEGO TC logo student guides: *Getting Started, Making Machines* and *Teaching the Turtle.*

Technic Control I

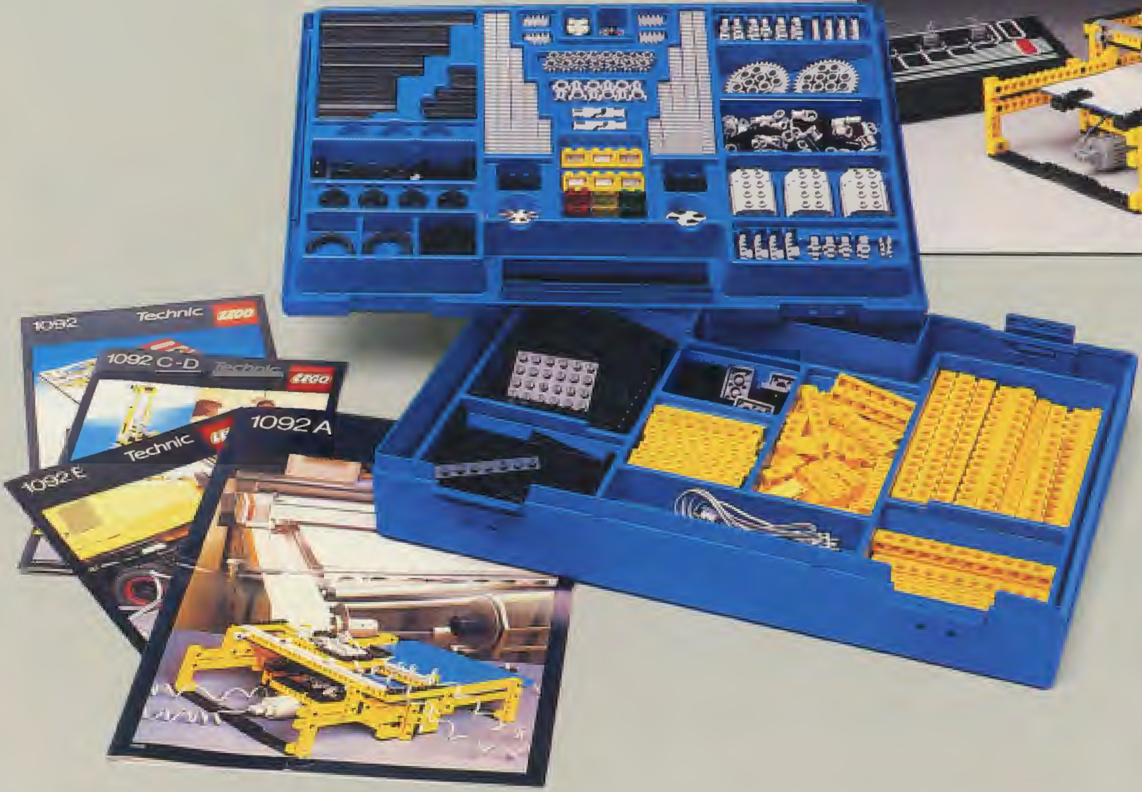
The Technic Control I set (item 1090) provides junior high and high school students with a hands-on learning environment to understand better the role of computers in today's technology. This set is designed for more advanced projects than the Technic Control 0 LEGO® TC logo set. Students can explore, investigate and apply mechanical principles, computer programming and fundamental control technology as used in real world situations. Experiments with programming and feedback enable students to form an in-depth understanding of control technology and information transfer.

Technic Control I, Item 1090, $138.00

- 404 elements, including two 4.5 volt motors, two optosensors and two counting wheels
- Step-by-step instructions to build five computer controlled models
- Storage tray with transparent lid
- Requires computer interface hardware and software available separately (see page 19). For Apple computers, either the Apple LEGO TC logo Starter Pack (item 951), or the combination of the Apple Slot Card Pack (item 955) and the Interface Box (item 9750). For MS-DOS computers, either the MS-DOS LEGO TC logo Starter Pack (item 966), or the combination of the MS-DOS Slot Card Pack (item 965) and the Interface Box (item 9750).

- *Computer controlled machines: robot arm, conveyor belt, automatic door, ferris wheel, washing machine*
- *Robotics, artificial intelligence, engineering, technology education*
- *Grades seven through twelve*
- *Designed for two students per set*

Technic Control II

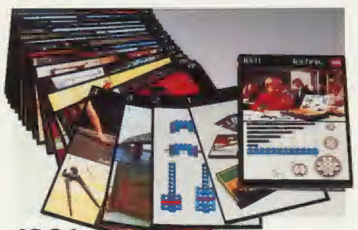


The Technic Control II set (item 1092) is especially appropriate for advanced junior high and high school students. The set focuses on the use of computerized control technology and advanced programming skills to design, build and operate scientific instruments for taking measurements and displaying data. Students can use these instruments to explore and investigate the physical world. They can apply the principles of instrument design and control technology to create even more complex instruments of their own.

- *Computer controlled machines: plotter, height measuring device, caliper, rotating base and two traffic lights*
- *Robotics, scientific measurement, technology education*
- *Grades seven through twelve*
- *Designed for two students per set*

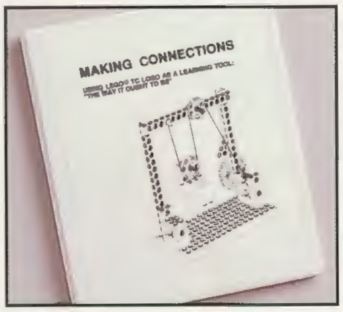
Technic Control II, Item 1092, $195.00

- 458 elements, including three 4.5 volt motors, two optosensors and two counting wheels
- Step-by-step instructions to build five computer controlled models
- Sturdy blue carrying case with two trays
- Requires computer interface hardware and software available separately (see page 19). For Apple computers, either the Apple LEGO® TC logo Starter Pack (item 951), or the combination of the Apple Slot Card Pack (item 955) and the Interface Box (item 9750). For MS-DOS computers, either the MS-DOS LEGO TC logo Starter Pack (item 966), or the combination of the MS-DOS Slot Card Pack (item 965) and the Interface Box (item 9750).


1031
$10.25 **TECHNIC I Activity Card Pack**

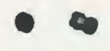
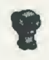
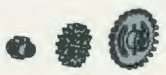
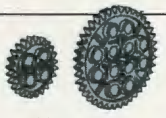
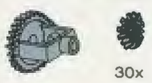
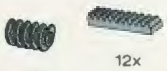
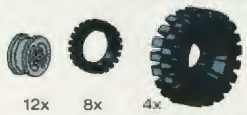
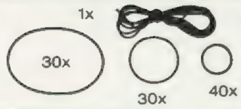
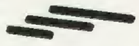
1033
$10.25 **TECHNIC II Activity Card Pack**

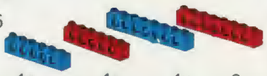
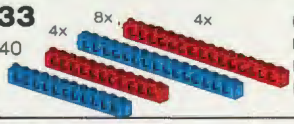
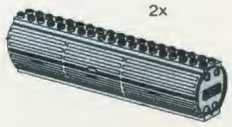
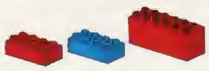
1072
$16.50 **304 extra bricks**



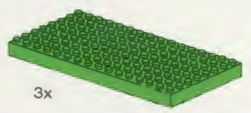
0069
$25.00 Making Connections is the product of an award-winning relationship between educators in St. Paul schools and the Science Museum of Minnesota. The book is a 160-page teacher's guide for LEGO TC logo with 34 creative lessons, four student-designed project ideas, a sample integrated unit for social studies and 16 physical science worksheets.

Spare part support services and technical specifications

1314	$7.25	 150x 60x	Stop bush – gray Small pulley – gray
1315	$4.25	 50x	Piston rod – gray
1316	$6.35	150x	Connector peg – gray
1317	$17.75	350x	Chain link – black
1318	$4.75	 50x 10x 8x	Gear wheel (8 teeth) – gray Gear wheel (16 teeth) – gray Crown wheel – gray
1319	$4.75	 14x 8x	Gear wheel (24 teeth) – gray Gear wheel (40 teeth) – gray
1320	$4.75	 2x 30x	Differential house – gray Bevel gear – gray
1321	$5.85	 10x 12x	Worm – gray Gear rack
1322	$7.15	12x 16x 4x	O-ring (tire) 20.2mm x 3.5mm – black Pulley wheel 24mm – gray Steering wheel – gray
1323	$4.75	 12x 8x 4x	Spoked hub Tire – black Tractor tire
1324	$7.25	 1x 30x 30x 40x	Rubber band, small – black Rubber band, medium – black Rubber band, large – black String – black
1325	$4.75	 30x 24x 14x	Cross axle, 4-studs long – Cross axle, 6-studs long – Cross axle, 8-studs long – black
1326	$4.25	14x	Cross axle, 10-studs long – black Cross axle, 12-studs long – black
1327	$3.20	12x 12x 12x 12x	Plate 1x3 studs – red Plate 1x3 studs – black Plate 1x4 studs – red Plate 1x4 studs – black

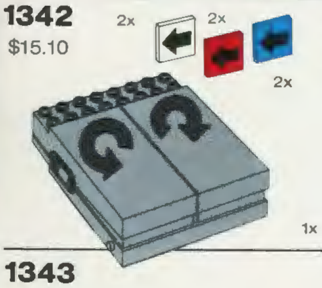
1328	$3.20	8x 8x 8x 8x	Plate 1x6 studs – red Plate 1x6 studs – black Plate 1x8 studs – red Plate 1x8 studs – black
1329	$3.20	8x 12x 8x	Plate 2x3 studs – black Plate 2x4 studs – black (with holes) Plate 2x4 studs – red (with holes)
1330	$4.25	8x 4x 8x	Plate 2x6 studs – red Plate 2x8 studs – red (with holes) Plate 2x8 studs – black (with holes)
1331	$7.40	24x 24x 24x 16x	Brick 1x2 studs – blue Brick 1x2 studs – red Brick 1x4 studs – blue Brick 1x4 studs – red
1332	$4.25	 4x 4x 4x 6x	Brick 1x6 studs – blue Brick 1x6 studs – red Brick 1x8 studs – blue Brick 1x8 studs – red
1333	$7.40	 4x 8x 4x 4x	Brick 1x12 studs – blue Brick 1x12 studs – red Brick 1x16 studs – blue Brick 1x16 studs – red
1334	$28.00	2x	Technic Motor 4.5 volts
1335	$11.90	 2x	Battery box
1336	$5.85	2x	Pole reverser switch for battery box
1337	$10.00	6x	Connecting lead
1338	$7.25	8x 12x 20x	Angle plate 2x2 studs – blue Swivel plate 1x2 studs – blue Swivel bearing 1x2 studs – blue Turntable – blue/gray
1339	$5.85	8x 4x 8x	Universal joint – gray Piston head Plate 2x2 studs round – red
1340	$4.75	 8x 8x 2x	Brick 2x4 studs – red Brick 2x4 studs – blue Weight element (6x2x2) studs – red

1341
$4.75



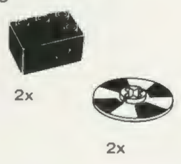
Building plate 8x16 – green

1342
$15.10



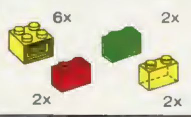
Control panel
Flat tile 2x2 studs – red
Flat tile 2x2 studs – blue
Flat tile 2x2 studs – white

1343
$41.25



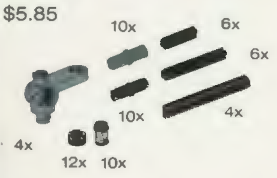
Optosensor
Counting disk

1344
$9.25



Light brick 2x2 studs – 4.5 volts
Transparent brick 1x2 studs – red yellow green

1345
$5.85

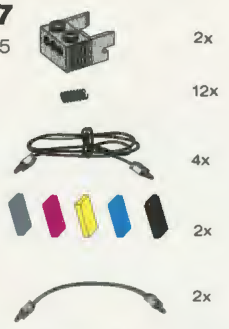


Pinion – gray
Connector peg with stud – gray
Nut – gray
Connector peg with cross axle – gray
Connector peg with friction – black
Cross axle, 2 studs long – black
Cross axle, 3 studs long – black
Cross axle with thread, 4 studs long – black

1346
$15.00

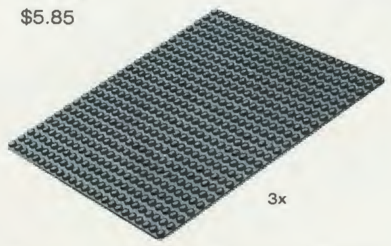
Touch sensor

1347
$17.75



Plug holder element
Spiral – black
3 m connecting cable
Flat tile 1x2 (5 different colors)
10 cm cable – gray

1348
$5.85

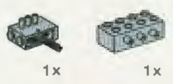


Building plate 24x24 – gray

5244
$3.09

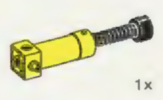
Broad chain link

5106
$3.69



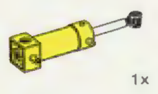
Two-way valve
Non-return valve

5107
$2.02



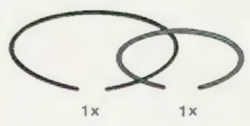
Pneumatic spring pump

5108
$2.02



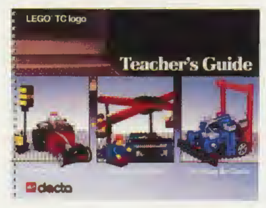
Pneumatic cylinder

5109
$2.02



Pneumatic tubing

957 $15.00 Apple®
967 $15.00 MS-DOS
LEGO TC logo
Reference Guide Pack



953 $12.50 Apple
963 $20.00 MS-DOS
LEGO TC logo
Teacher's Guide

954 $22.25 Apple
964 $22.25 MS-DOS
LEGO TC logo
Student Guide Pack

Technical Information

LEGO Dacta slot cards are available for the Apple IIe, Apple IIgs, and compatible computers, and the IBM PC and other MS-DOS computers. The cards are designed to transmit and receive data in a parallel fashion, and include onboard counters and timers.

The LEGO Dacta interface box works with all supported computers. It rests outside the computer and connects via a ribbon cable to a slot card installed inside the computer. A separate transformer supplies the voltage for lights and motors. The interface box is designed to a high standard of safety. For example, it is optically isolated from the computer power supply, thus protecting all users from the possibility of electrical injury. In addition, the circuits of the interface box cannot be damaged by students connecting motors, lights or sensors incorrectly. Finally, an onboard emergency STOP button provides students with a positive immediate method of turning off electrical power to all output slots if necessary.

The LEGO Dacta touch sensor consists of a special brick containing a spring-loaded on / off switch and a smooth protrusion which acts as a pushbutton.

The LEGO Dacta optosensor functions as a binary electromagnetic radiation sensor, and is sensitive to sharp changes between dark and bright conditions over a broad spectrum of radiation frequencies, including visible light and infrared radiation. For example, the sensor reacts to interruptions in a light beam shining on it. The optosensor also responds to the movement of a counting wheel by sensing changes in radiation emitted from an on-board infrared source and reflected by the black and white sectors of the wheel.

The LEGO Dacta 4.5 volt motor operates on direct current supplied from three "C" batteries or from the computer interface box. It uses approximately 0.1 amp and runs at approximately 6000 r.p.m. when operating with no load. When stalled, the motor uses approximately 1 amp.

Customer Support

Technical Support.
Having difficulty with a component of LEGO® Dacta equipment? By calling **1-800-527-8339**, you can consult with a trained LEGO Dacta representative and receive free online assistance.

Training.
Are you planning to present a workshop on LEGO Dacta materials? **Free** instructional modules are available. Would you like for LEGO Dacta to provide specialized instruction to a group? LEGO Dacta educational specialists are available for hands-on workshops and inservice training.

Grant Proposals.
Are you having difficulty obtaining funds for purchasing LEGO Dacta products? Consider writing a grant proposal to a local or national funding agency. Once you have identified a funding agency with an appropriate grant program, LEGO Dacta can provide free grant proposal preparation assistance, including program suggestions, review of draft proposals, writing assistance for equipment and training sections of proposals, and professional literature references. For first-time grant proposal writers, a **free** tip sheet is available on request.

Videocassette.
Would you like to see LEGO TC logo in action? LEGO Dacta has an introductory 16-minute videotape on LEGO TC logo available **free** to educators on a loan basis.

Literature.
Do you need extra copies of this brochure to distribute to your colleagues or to supplement handouts at a workshop or conference presentation? Additional copies are available **free.**

Spare Parts.
Are your students asking for additional motors, sensors or gears so they can build a more sophisticated project? Over 30 different spare part sets are available for purchase separately.

INVENT AMERICA!

LEGO Dacta is now a national corporate sponsor of INVENT AMERICA!, a nationwide educational program and invention competition for K-8 students. INVENT AMERICA! and LEGO Dacta are working together to help schools and teachers discover how LEGO products can be used as an integral part of the creative process. For more information on the INVENT AMERICA! invention competition, write to: INVENT AMERICA!, U. S. Patent Model Foundation, 510 King Street, Alexandria, VA 22314, or call (703) 684-1836.

LEGO dacta

LEGO Dacta
555 Taylor Road
Enfield, CT 06082
1-800-527-8339

could turn into the turtle shapes and colors that they describe, and be piloted around the screen.

- A display of piano keys could be used to sound out and automatically record single-note melodies.

The list of potential applications for a simple Logo interface seems endless.

Keep in Touch

The power of this Logo-to-graphics-pad connection made itself apparent when I designed a 12-disk set of materials for a local teacher of young physically and mentally challenged people. I imported many pictures from *Print Shop* graphics, and made a set of Logo Touch Window programs in four levels of difficulty that help users to refine gross motor movements. You are welcome to use these public domain programs, too. Send 12 blank disks (or 6 double-notched blank disks) in a self-addressed, sufficiently stamped disk mailer to the address at the end of the article. I will copy the materials onto the disks and return them to you.

If you are curious about how to use *Print Shop* graphics in Logo programs, or Logo pictures in *Print Shop* creations, please refer to my September 1988 "Logo LinX" article, "Secular Conversions," in *Logo Exchange*.

If you are interested in learning more about controlling peripheral devices such as speech synthesizers and videodisc players with Logo, or accessing other input devices such as temperature or light sensors with Logo, the following articles may be helpful.

References

Barclay, T. (1987). To graph a probe–Logo in the science lab. *The Computing Teacher*, *14*(8), 30-32.

Bull, G., Bull, G., Lough, T., Harris, J. and Wissick, C. (1988). Logo connections: An open architecture for Logo. *Logo Exchange*, *6*(8), 13-16.

Bull, G. & Cochran, P. (1987). Science and sensors. *Logo Exchange*, *5*(5), 9-11.

Bull, G. & Cochran, P. (1987). Science and sensors II. *Logo Exchange*, *5*(6), 9-11.

Cochran, P. & Bull, G. (1986). Touch tools. *Logo Exchange*, *5*(4), 18-20.

Harris, J. (1988). Secular conversions. *Logo Exchange*, *7*(1), pages undetermined at press time.

Product Information

"Animation Station," Suncom, Inc, $99.95
"KoalaPad," Koala Technologies, $125.00
"Touch Window," Personal Touch Corp., $199.00

Note: A previous version of this article appeared as the November 1988 "Logo Center" column in *The Computing Teacher*.

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

About the Cover

The perspective drawings to this month's cover were done by students of Orlando Mihich, a science teacher at Junior High School #118 in New York City. She writes

> My kids got interested in perspective work after seeing my earth science drawings on the board, and after reviewing a book on Giorgio de Chirico. I explained the basic concepts of perspective drawing mainly during lunch hours and after school. Using the **seth towards []**, they drew lines looking towards a horizon, **make "x pos** helped them **setpos :x** for shading and so on. They came out with some quite interesting art work. The...sphere was earlier used to represent atoms in a crystal lattice; here it is used to add a round, soft shape to otherwise straight towers and pyramids. The distant LogoWriter train is the mysterious, metaphysical part of De Chirico's world itself.

DeChirico was done by Michael Toribio, 9th grade; Morning Meditation was done by John Deveaux, 8th grade; and Yutz kin was done by Alex Acevedo, grade 9.

MathWorlds

edited by
A. J. (Sandy) Dawson

Given that this issue of *LX* is devoted, in some sense, to experts, it seems particularly appropriate for Jim King to make his second appearance here this year. The Babylonian Turtle that Jim introduces below challenges Logo and mathematics experts to solve several cases involving wayward turtles. Read on, but be sure to stop along the way to exercise your Sherlock Holmes capabilities in finding the solution to Jim's Logo Mysteries.

The Case of the Babylonian Turtle: A Logo Mystery

by Jim King

A series of mysterious events has taken place:

- A usually reliable turtle sets off to draw a regular polygon but does not return to its starting point.
- A turtle sets off for a spot on the opposite side of the screen, but misses its destination.
- A scientific turtle conducts experiments to find the value of pi, but the results do not agree with theoretical predictions.

N-gons and the Babylonians

When we begin to draw polygons in turtle geometry, we are naturally attracted to polygons like the triangle, the square, the pentagon, the hexagon, and the octagon, all of which can be drawn by a repeated sequence of FORWARDs and turns by an integer number of degrees: 120, 90, 72, 60, 45. Many other regular polygons with N sides, or N-gons, require a turn that is not an integer number of degrees: the heptagon (the 7-gon, which requires a turn of 360/7 = 51 3/7 degrees), the 11-gon, and the 16-gon are examples. Since the turn required to draw an N-gon is the quotient of 360 divided by the number of sides N, the turn is an integer only when N divides 360 without remainder.

From the mathematical point of view, it is important to note that this distinction between N-gons that have integer turns and those that do not is an artificial one rather than a basic phenomenon of geometry. This artifact depends on how we measure angles. We still follow the scheme invented by the Babylonians; they measured angles by dividing a circle, or a complete turn, up into 360 parts called degrees. If the Babylonians had chosen to divide the circle into 350 units called zorts, then the heptagon would be drawn with a turn of 50 zorts but the triangle would require a turn of 116 2/3 zorts. So the division of N-gons into "nice" ones with integer turns and "not-so-nice" ones with non-integer turns would have given completely different results.

Thus, if we write a Logo procedure like this, it should draw a polygon for any positive integer input, not just the ones that divide 360 evenly.

```
TO NGON :N :SIDE
REPEAT :N [FD :SIDE RT 360/:N]
END
```

However, if you actually try this with certain versions of Logo, the turtle does not come back to where it started. For example, if the turtle is at the home position and if you draw a heptagon by NGON 7 60 with IBM Logo, the final position of the turtle is [0.52 0.421]. Why is this so?

You can't get there from here

Here is the second mystery. Suppose the turtle is at [0 0]. The following sequence of commands should move the turtle to [50 120] and print 50 120; but in IBM Logo and some others, what is printed is 48.698 120.532. (Notice that the distance from [0 0] to [50 120] is the square root of $50^2 + 120^2 = 130$, by the distance formula in the (x,y) plane.)

```
HOME
SETH TOWARDS [50 120]
FORWARD 130
PRINT POS
```

Compare this with the result of HOME SETPOS [50 120]. The difference is quite visible on the screen. Why does this happen? Does it happen with the version of Logo that you use?

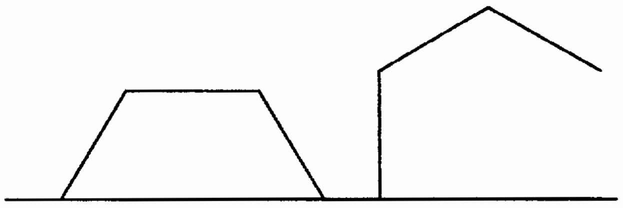
Semicircles and pi

The following procedure was suggested as a way of estimating the value of pi.

```
TO EST.PI :N :SIDE
PU
SETPOS [-125 0]
SETH 0
PD
SEMI :N :SIDE
PRINT SENTENCE [ESTIMATE OF PI = CIRCUM/
   DIAM = ]  2 * :N * :SIDE/(XCOR + 125)
END

TO SEMI :N :SIDE
REPEAT :N [RT 90/:N FD :SIDE RT 90/:N]
END
```

The idea is that the turtle should start at [-125 0], pointing upward, and draw a "Logo semicircle," half a 2N-gon. The semicircle is drawn by REPEAT :N [RIGHT 90/:N FORWARD :SIDE RIGHT 90/:N] rather than REPEAT :N [FORWARD :SIDE RIGHT 180/:N] so that the half 2N-gon will end up on the x-axis where it began. The difference between these two versions of the half 2N-gon is illustrated in the figure below for N = 3.



It was understood that round-off error would become a problem for very large values of :N, but the idea was to experiment to see which value of :N would give the best estimate for pi.

What happened, however, is that the Logo semicircle was tilted for some values of :N. For example, using LogoWriter 2.0 for the Apple II, with :N = 4 and :SIDE = 80, the final y-coordinate of the turtle was -1.8243. This is a rather large error for 4 steps. On the other hand EST.PI 5 80 results in a y-coordinate of 0 as it should, although one would expect the round-off error to be about the same for both :N = 4 and :N = 5 since the number of sides is about the same.

Round off the usual suspects

Your first reaction to this may be, as mine was, "round-off error." Round-off error is the error that inevitably occurs when we try to compute with numbers that have infinite decimal expansions. Since the computer can only work with a finite number of decimal places, every arithmetic operation yields a small error; after many operations, this error can become quite significant.

But this does not explain the mystery here. First of all, the round-off error for integer and non-integer angles should be about the same; for even when the number of degrees in the turn is an integer, the (x,y) coordinates of the positions are irrational numbers subject to round-off error. Put the instruction PR POS inside the REPEAT of NGON to see this.

Also, Logo works using a fairly large number of significant digits. In the heptagon example, for instance, if at each step there were a round-off error in the sixth decimal place, an error of less than 0.00001, then after seven steps the error could not be more than 0.00007. This could not account for the large observed error.

Babylonian turtles

It turns out that the problem is that, for some brands of Logo, the turtles are Babylonian. If you ask them the HEADING, they will tell you to several decimal places, but when you tell then FORWARD, they round the heading to the nearest integer before going FD. In other words, SETH 0 RIGHT 42 FORWARD 100 and SETH 0 RIGHT 42.4 FORWARD 100 give the same position. "Position" here does not only mean the relatively crude measure of position given by the pixels on the screen, but also the value of POS, which Logo uses to determine where the turtle goes next if it is given another FORWARD command.

Interestingly enough, the heading is not rounded, so the sequence of commands SETH 0 RIGHT 42.4 FORWARD 100 RIGHT 12.4 FORWARD 10 draws the same path as SETH 0 RIGHT 42 FORWARD 100 RIGHT 13 FORWARD 10 even though 12.4 does not round to 13, because for the first FORWARD, the heading is 42.4, which is rounded to 42, but for the second, the heading is 54.8, which is rounded to 55; it takes a RIGHT 13 to turn from a heading of 42 to a heading of 55.

This explains what happened to the heptagon. The value of the heading that the turtle is using for the FORWARD is much less accurate than we thought, with no significant figures to the right of the decimal point. Once we realize this, it is not surprising that the error is as large as it is.

The second problem with SETH TOWARDS has the same explanation. The turtle is going in the direction of an integer heading, not the true one.

In drawing the semicircle in the pi estimation experiment, if there is an error in the initial turn, the whole semicircle is rotated by that amount (and the semicircle is not really regular, since its angles are not all the same).

Why are some turtles Babylonian?

I have not asked the programmers who wrote various versions of Logo, but it seems pretty clear why one might choose to make the turtle Babylonian instead of accurate—speed. Computing where the turtle will end up after a FD command involves computing sines and cosines, which is a fairly slow process. If you only have 360 angles to deal with, you can store the values of the sine and cosine in a table and look them up without having to compute them each time. Since the Logo turtle can seem pretty slow, this is one way to get it moving along at a brisker pace, but at the price of accuracy.

How can I get an accurate turtle?

Brian Silverman has created a very interesting microworld called Broken Logo. In this microworld he alters the behavior of the turtle commands so that the turtle behaves differently from what one expects. Silverman then challenges the microworld explorer to fix up the turtle so it behaves properly again.

Amusingly enough, many versions of Logo, unbeknownst to its users, are already a form of broken Logo. How can we fix the Babylonian turtle so that it becomes an accurate turtle? Since Logo is extensible, we just define a command CFD (for Correct FD).

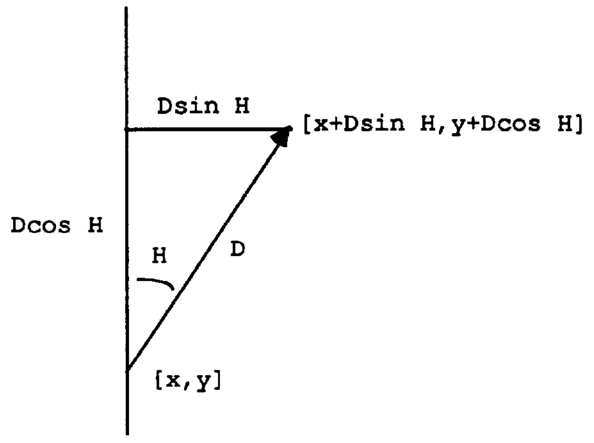
```
TO CFD :DIS
SETPOS LIST (XCOR + :DIS * SIN HEADING)
    (YCOR + :DIS * COS HEADING)
END
```

```
TO CBK :DIS
CFD -:DIS
END
```

If your Logo has SETXY but not SETPOS, then define CFD using this line in the definition:

```
SETXY (XCOR + :DIS * SIN HEADING) (YCOR +
    :DIS * COS HEADING)
```

We can see that this is the correct new position from the following figure. Unfortunately, when this CFD command is computed using an interpreted Logo, it is much slower than the usual FORWARD. (Perhaps it would be better named SFD for SLOW FD.)



One thing CFD is good for is to test turtles for Babylonian tendencies. Get out your favorite Logo and see whether FD and CFD have approximately the same result.

Are Babylonian turtles a serious problem?

For most purposes, the errors introduced by Babylonian turtles are not serious. People tend to use a lot of integer turns in Logo, and errors do not appear in this case. Also, unless the computer has good graphics, some of the errors are not visible on the screen. CFD is so slow that one wants to avoid using it as much as possible. Nevertheless, it is good to be aware of what is really is going on with one's turtle in case the turtle seems to develop mysterious ailments. In my own case, for a certain project I wrote a set of procedures that used SET TOWARDS and then FD many times and was mystified by the strange behavior of the turtle until I discovered its guilty secret.

It would be interesting to make a list of which versions of Logo have Babylonian turtles and which do not. As I have indicated, IBM Logo and LogoWriter 2.0 for the Apple II fall into the Babylonian camp. On the other hand, Object Logo and Terrapin Mac Logo do not.

As Logo continues to grow up and as computers get faster and more powerful, it seems a better choice to be given an accurate turtle instead of a Babylonian one. In any event, designers of Logo would do their users a service if they would document what their software really does.

Puzzles and Projects

- Even though 16 does not divide 360 evenly, the 16-gon drawn by a Babylonian turtle with the instruction NGON 16 60 closes up exactly (i.e., the final position of the turtle is [0 0]. Why is this so? Can you find other cases of this nature?
- If your version of Logo has an accurate turtle rather than a Babylonian one, how can you simulate a Babylonian turtle? In other words, define a procedure BFD :DIS that will cause the turtle to do what a Babylonian turtle would do when given the instruction FORWARD.
- Carry out the pi-estimation experiment both with FD and with CFD. What are your results?
- Define a distorted version of CFD this way:

```
TO DFD :DIS
SETPOS LIST (XCOR + 2 * :DIS *
    SIN HEADING) (YCOR + :DIS * COS
    HEADING)
END
```

Compare what happens when you draw a Logo circle (an N-gon with large N) using regular FD, or by substituting CFD or DFD for the FD in the definition of NGON. What happens when you replace the 2 in the definition of DFD with other numbers? What happens if you put a number in a comparable place in the second coordinate of the SETPOS?

Note: IBM (IBM Corp.), IBM Logo (IBM Corp.), Apple II (now LCSI Logo II, LCSI), LogoWriter (Logo Computer Systems), Terrapin Mac Logo (Terrapin, Inc.), Object Logo (Apple Computer Corp.) are trademarks.

James King
Department of Mathematics GN-50
University of Washington
Seattle, WA 98195 or via E-mail as
king@math.washington.edu

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

Three-Dimensional Logo

by Horacio C. Reggini

Introduction

I believe that the use of computers as a medium of self-expression—a medium for exploring, testing, creating, realizing, understanding—is the main reason for introducing computers in education. In this article, I use the computer for the description and generation of three-dimensional shapes.

We can give students an active role as "shape-builders," allowing them an intellectual excursion into the field of creation and manipulation of three-dimensional shapes. The aim is to stimulate and encourage the exploration of the structure and composition of geometrical shapes. Model building with all kinds of blocks is recognized by educators as a valuable part of education, but unfortunatelly it is usually restricted to young children. Model building — both hands-on and conceptual — should be a central part of educational goals. Computer model building with 3D-Logo shares important characteristics of both aspects, and leads students to important questions about the geometrical principles of forms from nature and from human design. By teaching the computer how to produce three-dimensional objects, students may also achieve greater understanding of the aesthetics and complexity of shapes in space.

Creation of three-dimensional shapes

The perception of the space around us is a concept difficult to grasp. For a child, learning to know the spatial world outside his own skin is an experience that lasts many years.

There are many ways of describing an object in space. We know that the global geography of space can be inferred just from hints about which pairs of points lie near one another. A similar concept underlies the simple method I explain here: an object can be described and generated by starting from any of its points and demonstrating the movements necessary to trace its contours. This geometrical description makes no reference to any external element, but it is intrinsic to the object itself.

Let us imagine our hand, open and extended, following the edges of a three-dimensional object. The command TRAVEL designates the movement of the hand along the direction of the fingers. I call VEER, PITCH, and ROLL the following rotations of the hand.

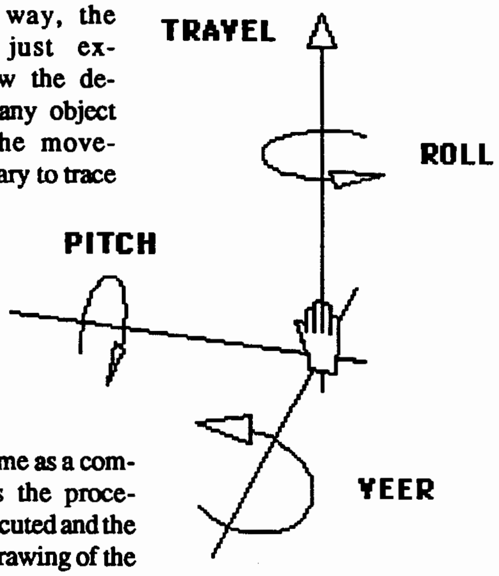
VEER is the rotation in the plane of the palm of the hand; any two-dimensional shape can be totally defined, therefore, just with successive commands TRAVEL and VEER.

PITCH is the rotation similar to turning the hand down or up by deflecting the wrist.

ROLL is the rotation corresponding to a corkscrew movement.

If we compare our hand, placed in a horizontal position, to a ship, then VEER will correspond to a change of heading, PITCH to the stern-bow oscillation, and ROLL to the port-starboard movement. In international aeronautical language, the movement corresponding to VEER is called YAW, while the names of the other movements are the same.

In this way, the commands just explained allow the description of any object following the movements necessary to trace its contours. This description is stored in the computer as a program or procedure with a name. Using this name as a command causes the procedure to be executed and the perspective drawing of the described object is displayed on the computer screen.



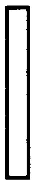
The commands TRAVEL and VEER are equivalent to the well known FORWARD and LEFT of the classical two-dimensional Logo. This intrinsic, natural way of working out geometrical problems has been called "turtle geometry" or "turtle language," because lines are made as if a turtle executed them. When making the turtle move we externalize our idea about how to create a form. Moving according to the itinerary imposed by the procedure, the turtle leaves a trail or a mark, thus creating the desired shape.

In three-dimensional Logo the concept of corporal sintonicity, one of the powerful ideas elaborated by Papert for two-dimensional Logo, is still valid. I have talked about the movements of the hand instead of talking about a turtle, which in space, would have to be an aerial or aquatic turtle, but the idea of manipulating an "object-to-think-with" remains. We learn to think in a rigorous but, at the same time, corporal and intuitive way, and we use mathematics like a natural language.

Some elegant and graceful shapes

The following pages show several simple examples of how to define objects using three-dimensional Logo. To begin, let us first define a small rectangular piece 8 x 64, called PLATE, which will be the essential building block to create more complex designs:

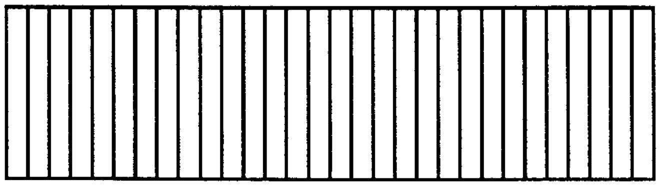
```
TO PLATE
PU
TRAVEL -32
PD
REPEAT 2 [TRAVEL 64 VEER 90 TRAVEL 8 VEER
    90]
PU
TRAVEL 32
PD
END
```



PLATE

The PLATE is generated starting from the middle point of one of the longer sides of the rectangle and ending at the same point. This point —as we will see later—will be a kind of joint for successive adjacent PLATEs. We can think of PLATE as a thin flat narrow piece of metal. As a jeweller, we will assemble them, building up different designs.

1. Let us combine several PLATEs, shaping a kind of strip on the plane:



PLANAR.RIBBON

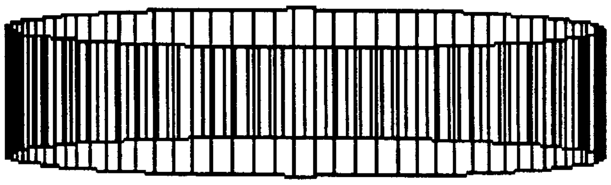
```
TO PLANAR.RIBBON
REPEAT 30 [PLATE MOVE.1]
END

TO MOVE.1
PU
VEER 90
TRAVEL 8
VEER -90
PD
END
```

MOVE.1 is the procedure that tells the procedure PLANAR.RIBBON how to assemble the PLATEs.

Note that the procedure MOVE.1 moves the turtle 8 units to the left without leaving any trail, because we have moved it in the PENUP state (abbreviated PU). With the command PENDOWN (abbreviated PD), the turtle again leaves a trail while moving.

2. Suppose that now we add a ROLL movement to MOVE.1, thus defining a new procedure for joining PLATEs which we call MOVE.2. So, we create a kind of round strip, no longer on the plane but in the space. The resulting object is similar to a cylinder or circular band:



CIRCULAR.RIBBON

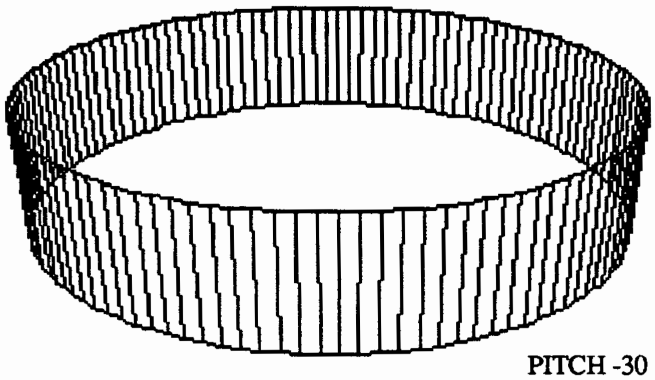
```
TO CIRCULAR.RIBBON
REPEAT 120 [PLATE MOVE.2]
END

TO MOVE.2
MOVE.1
ROLL -3
END
```

Note that the procedure MOVE.2 adds to the procedure MOVE.1 a ROLL angle 3 to the right. So as to complete a circle, we REPEAT 120 times the commands [PLATE MOVE.2].

The construction procedure CIRCULAR.RIBBON creates the shape as a sum of PLATEs. We suppose that each PLATE is of a transparent material with opaque visible edges. Consequently, the obtained image of the CIRCULAR.RIBBON is a kind of a wire-frame drawing, where all the lines are visible. In some cases, we could easily avoid the drawing of hidden lines as if the faces of the object were non-transparent.

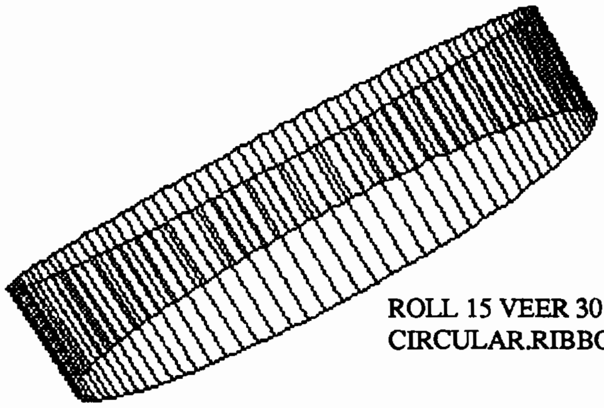
If we order PITCH before CIRCULAR.RIBBON, the following image will result:



PITCH -30
CIRCULAR.RIBBON

Notice how the command PITCH -30 before CIRCULAR.RIBBON produces an image of the object as seen from above with a 30 degree tilt. This happens because, in 3D-Logo, the starting position and heading of the hand define, in relation to the viewpoint situated in front of the center of the screen, the image of the three-dimensional object that the procedure creates. When we begin with 3D-Logo, we suppose that the hand rests on the center of the screen with the fingers pointing up. It is the classical initial position of the turtle at the beginning of any activity.

So any movement given to the turtle before commanding the name of an object will affect the perspective of the object drawn by the computer. Thus, we can obtain different perspectives of the same object from different points of view. For instance:



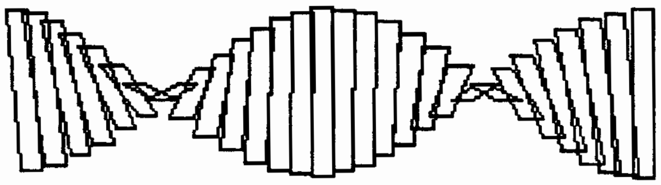
ROLL 15 VEER 30
CIRCULAR.RIBBON

3. Now, let us define MOVE.3 which joins contiguous pieces including a PITCH rotation in between. The group of PLATEs joined with MOVE.3 shapes, in this case, a kind of twisted strip:

```
TO TWISTED.RIBBON
REPEAT 30 [PLATE MOVE.3]
END
```

```
TO MOVE.3
MOVE.1
PITCH 12
END
```

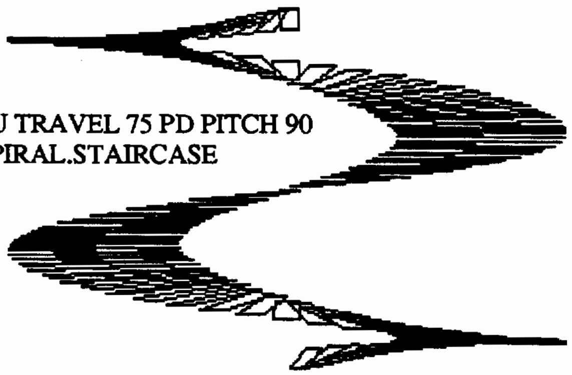
TWISTED.RIBBON



The twisting degree of the ribbon is given by the relation between the angle given by PITCH 12 and the displacement given by TRAVEL 8 in the procedure MOVE.1. A greater value of the angle in relation to the value of the displacement will result in a more twisted ribbon.

It is interesting to realize how small changes in the MOVE procedures produce large changes in the results at the end, just as subtle changes in the genetic code do. A computer program can be considered as a "society of procedures," after Marvin Minsky's terminology. Making a large program means assembling many little parts and processes, each of them made up of very simple instructions. It is remarkable to see how large entities do not rely directly on the little parts that form them. What is important here is the whole, the way the parts affect each other, and not what they are in themselves.

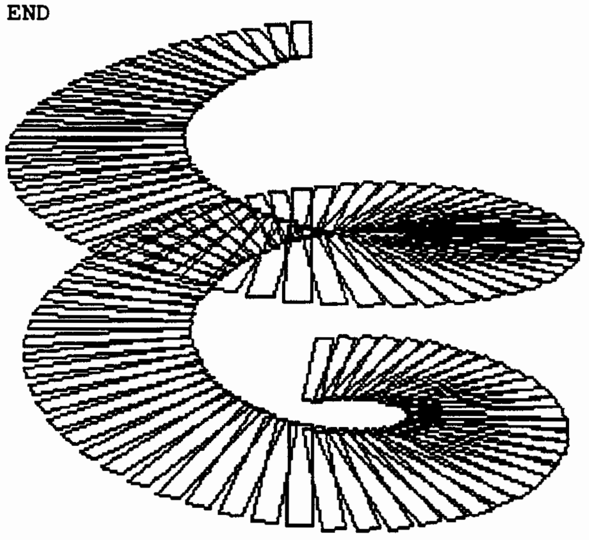
4. In this section, let us place PLATEs combining several movements, with the intention of creating a shape similar to a spiral staircase. We use PLATE as the tread, stepping upwards by means of a ROLL movement before a translation given by the MOVE.1 procedure. Then, we impose a ROLL movement of inverse sign, in order to whirl around the axis of the staircase.



PU TRAVEL 75 PD PITCH 90
SPIRAL.STAIRCASE

```
TO SPIRAL.STAIRCASE
REPEAT 120 [PLATE MOVE.4]
END

TO MOVE.4
ROLL -9
MOVE.1
ROLL 9
VEER 6
END
```



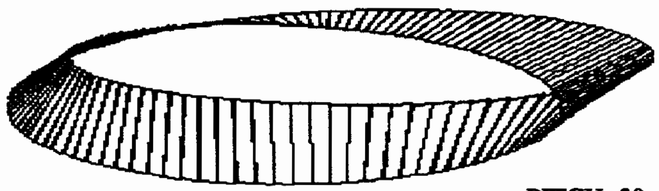
PU TRAVEL 75
PD
PITCH 70
SPIRAL.STAIRCASE

The shapes displayed above could encourage some readers to build helicoidal ribbons similar to DNA (deoxyribonuclei acid) structures.

5. Let us insist on joining PLATEs, mixing both ROLL and PITCH commands. If, when the last PLATE meets the first, the PITCHed angle totals 180 degrees, we create a Möbius strip.

The Möbius strip, named after August F. Möbius, is a surface with only one side, formed by giving a half twist to a narrow, rectangular strip of paper or any other material, and then pasting its two ends together.

We can immediately create a procedure called MÖBIUS by using procedure MOVE.2 which ROLLs an angle 3 (360/120). As the recursive procedure MÖBIUS is executed 120 times, consequently the relative PITCH angle between contiguous PLATEs must be 1.5 (180/120) in order to produce one half-twist. In this way, the total rotation is 180 for the sum of the relative PITCH angles and 360 for the sum of the ROLL angles.



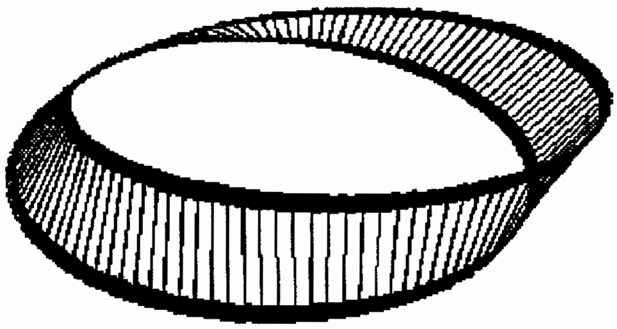
PITCH -30
MÖBIUS 0

```
TO MÖBIUS :TILT
IF :TILT = 180 [STOP]
PITCH :TILT PLATE PITCH -:TILT MOVE.2
MÖBIUS :TILT + 1.5
END
```

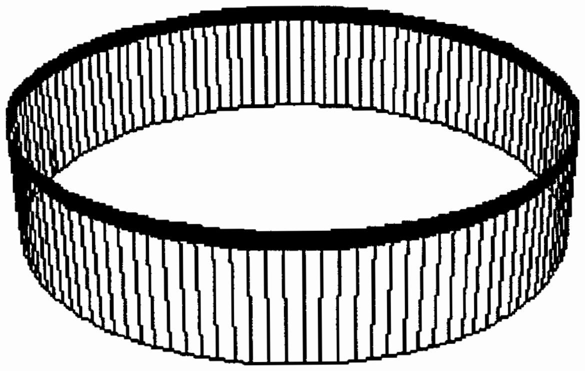
As it is known, the MÖBIUS strip has only one bounding edge. We can verify this by coloring its contour as we go over it. One way could be to define a procedure MARK which would paint a black square at one extreme of the PLATE. Then, by means of the procedure MARK.BORDER, we could repeat MARK all along the border:

```
TO MARK
PU
TRAVEL 24
PD
SQUARE 8
PAINT.SQUARE
PU
TRAVEL -24
PD
END

TO MARK.BORDER :TILT
IF :TILT = 360 [STOP]
PITCH :TILT MARK PITCH -:TILT MOVE.2
MARK.BORDER :TILT + 1.5
END
```



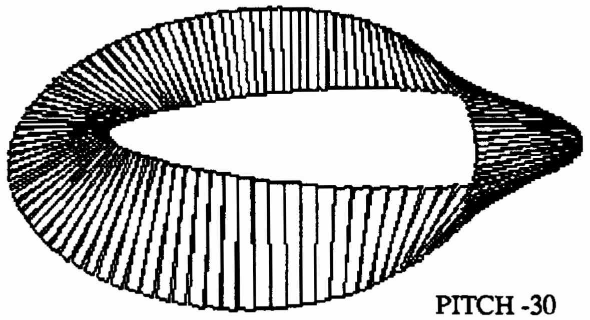
Compare the last figure with the following one corresponding to the CIRCULAR.RIBBON –which is an ordinary surface with two sides and two boundary edges. We have gone over and colored one of its two borders in a similar way:



In the MÖBIUS procedure, we have twisted one end of the strip 180 degrees before joining it with the other end. It is easy to introduce a change in the MÖBIUS procedure, including as input the number of :HALF.TWISTS of the strip:

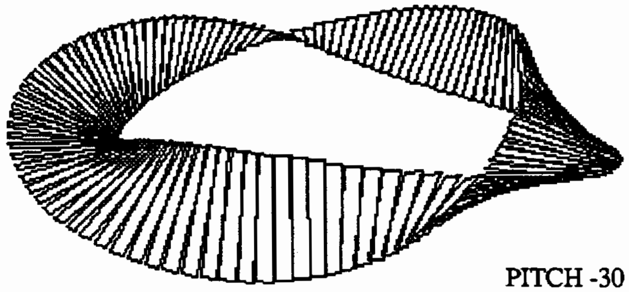
```
TO HIGH.ORDER.MÖBIUS :TILT :HALF.TWISTS
IF :TILT = 180 * :HALF.TWISTS [STOP]
PITCH :TILT PLATE PITCH -:TILT MOVE.2
HIGH.ORDER.MÖBIUS :TILT + 1.5 *
    :HALF.TWISTS :HALF.TWISTS
END
```

HIGH.ORDER.MÖBIUS is a variable procedure for building Möbius strips of :HALF.TWISTS-th order. When :HALF.TWISTS is 2, we get a full twist:



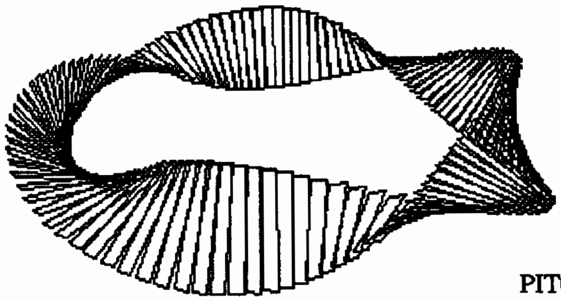
PITCH -30
HIGH.ORDER.MÖBIUS 0 2

When :HALF.TWISTS is an odd number, the surface is one-sided and possesses a single boundary curve which is knotted when :HALF.TWISTS is greater or equal to 3. For instance, if we command three :HALF.TWISTS, we obtain:



PITCH -30
HIGH.ORDER.MÖBIUS 0 3

When :HALF.TWISTS is an even number we obtain strips with two sides and two linked boundaries, like the one we obtained when :HALF.TWISTS was 2. Finally, let us see the resulting image when :HALF.TWISTS is 4:



PITCH -30
HIGH.ORDER.MÖBIUS 0 4

Final remarks

The way of describing a shape with 3D-Logo is different from other methods based, for example, on the individualization of the coordinates of particular points of the objects, which are related to criteria of extrinsic geometry. The definition of an object made with 3D-Logo is not only straightforward but it also bears relevant information about the object itself. The shape is characterized by its intrinsic geometry; that is, by the distances, angles, faces, and connections of its parts.

We build forms studying their basic structural units and how to fit them together. A space definition arises, as in most Logo projects, from a combination of simple subprocedures. The different subprocedures used to assemble the various parts are comprehensible as they correspond to the natural movements one should follow when building up a real object. The properties of a shape can be deeply studied and experimented with. Any object definition can be conveniently retrieved and combined with new designs.

It is worthwhile to realize that 3D-Logo methodology focuses not on the representation of the object on the screen, but rather, and this is the essential point, on the description and creation of the object in space. This description, formalized by the respective procedure, is similar to the artisanal building of a shape using the basic commands explained above. We can imagine that we have a kind of chisel that we use to carve the object in space. Each part of the object is created and joined by successive movements individualized by spatial commands. The process is also a kind of "robot language," as it can be used to guide and control robotic devices or to produce computer generated holograms.

There are other computer languages and systems that can help students to build three-dimensional computer constructions, but what makes 3D-Logo oustanding are its simplicity and natural approach to geometry. A person is not forced to assimilate new and complicated methods or terminology just to use the computer as a versatile tool.

Moreover, the 3D-Logo system can be immediately implemented in most of the existing computers using Logo. It is only necessary to load the correspondent procedures that define the 3D-Logo commands.

Once an object is defined by means of a procedure, we can call it by its name and we can manipulate it as if we had it in our hands. This becomes specially vivid when displaying the perspective of the object on the screen. When a procedure is executed, it is instructive and fascinating to watch how the object is created along successive stages by the corresponding subprocedures. Thus computers become active learning tools, providing students with an opportunity to feel the emotion and joy of the creative act.

In a revitalized geometry curriculum, a more central role should be given to the study of three-dimensional forms. 3D-Logo can significantly contribute to that purpose.

References

Abelson, Harold and diSessa, Andrea A. (1981). *Turtle Geometry*, Cambridge, MA: MIT Press.

Coxeter, H. S. M. (1969). *Introduction to Geometry*. New York: John Wiley and Sons.

Minsky, M. (1987). *The Society of Mind*. New York: Simon & Schuster.

Papert, S. (1980) *Mindstorms*. New York: Basic Books, New York, 1980.

Reggini, H. C. (1985, July). *Three-dimensional space with Logo*. Paper presented at Plenary Session IV, Logo-85 Conference, Cambridge, MA. Published in *MICRO-MATH*, 2(1), Spring, 1986.

Reggini, H. C. (1985). *Ideas y formas, explorando el espacio con Logo*. Buenos Aires: Ediciones Galápago. (French translation: (1986). *Logo dans l'espace*. Cedic/ Nathan: Paris; Italian translation: (1987). *Idee e forme*. Roma: Sisco Sistemi Cognitivi.)

Reggini, H. C. (1986). Towards and artisanal use of computers, their application to the design and study of three-dimensional forms. *Proceedings of the Third International Logo Conference*, Cambridge, MA.

Reggini, H. C. (1986). *Exploring 3-dimensional space with Logo* (Logo Memo 102). Cambridge, MA: MIT.

Reggini, H. C. (1986). *3-dimensional Logo commands and procedures* (Logo Memo 103). Cambridge, MA: MIT.

Reggini, H. C. (1986). *Projects with 3-dimensional Logo* (Logo Memo 104). Cambridge, MA: MIT.

Reggini, H. C. (1987). Creaci6n y representaci6n de formas tridemensionales. *Anales del la Academia Nacional de Ciencias Exactas*. Buenos Aires.

Horacio Reginni
Buenos Aires, Argentina

Logo and Company

Building a Turtle in *HyperCard*: "Mock Turtle"

by Glen L. Bull and Gina L. Bull

HyperCard was developed by Bill Atkinson. Bill Atkinson was also the developer of *MacPaint*, a program responsible for much of the early interest in the Macintosh, and *QuickDraw*, the native graphics language of the Macintosh. *HyperCard* was initially intended to be an educational authoring system for the Macintosh. In fact, Bill Atkinson mentions Logo as one of the programming languages that influenced the development of *HyperCard*. There were, of course, many other sources of inspiration, but it is safe to say that anyone who uses Logo will also find *HyperCard* to be a comfortable environment in which to work.

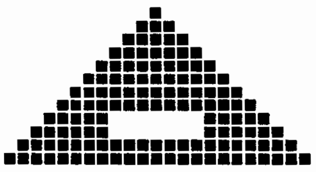
Given the early history of *HyperCard*, and Bill Atkinson's background in the development of innovative graphics applications, it is somewhat surprising that *HyperCard* does not include turtle graphics. Turtle graphics have spread from Logo to many other languages, including Pascal and BASIC. Although *HyperCard* has very strong paint tools for development of freehand illustrations, Logo users will be less comfortable with its tools for creation of graphics programs, since they are based on Cartesian coordinates rather than turtle geometry.

Building the Turtle

HyperCard provides all the tools required to build a turtle, however. In this column we will build the basic turtle and develop the commands required to move it around the screen, such as RIGHT, LEFT, FORWARD, and BACK. In the previous two columns we showed you how to create a *HyperCard* button called compass, and how to write commands such as UP and DOWN to move the button around the screen. The button named "Compass" looked like this.

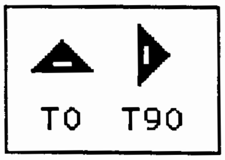
In this column we will build a turtle shape that will replace the compass. In the same way that some versions of Logo provide a shape editor that allows the shape of the turtle to be altered, the *icon* displayed on a *HyperCard* button also can be changed. Some icons, such as the compass shape, are supplied with *HyperCard*. It is also possible to create additional icons that can be used with *HyperCard* — up to 32,000 more.

There are several ways in which this can be done. The turtle shape could be created using the *Fat Bits* option in the *HyperCard* paint tools, and then converted to an icon with a shareware program such as *Icon Maker*. In this instance we used a commercial program called *Icon Factory* to create the turtle shapes. (Sources for these programs are listed at the end of the column.)

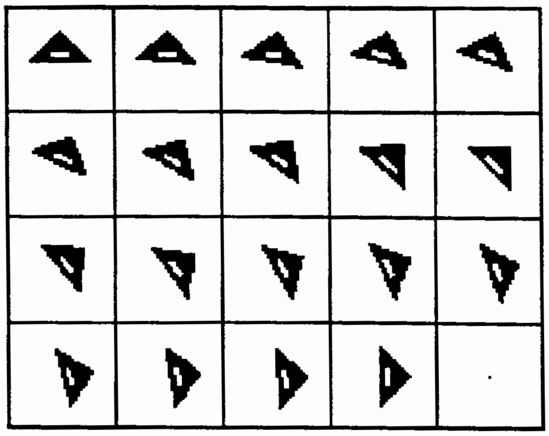


Note: If you do not have an icon editing program at this time, you will be able to use one of the existing icons built into *HyperCard* as a "turtle substitute." Information on substitution of an existing *HyperCard* icon for the turtle will be provided in the next section.

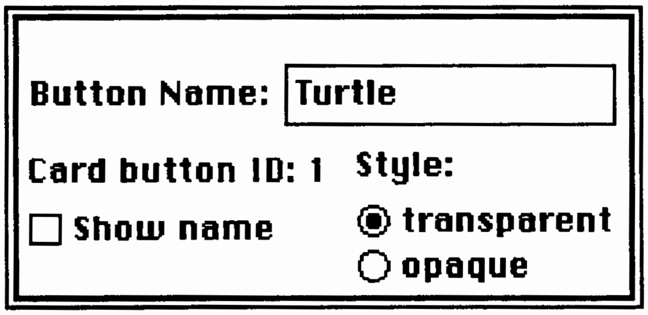
To create the illusion that the turtle is turning, it is necessary to create different turtle shapes for different directions. It is not necessary to create a different turtle shape for each degree of rotation, but only for every 5 or 10 degrees. If you look at the turtle in your version of Logo closely, the chances are that you will notice that the shape does not change when you type "Right 1." It only changes when you type "Right 5" or "Right 10." We looked at several versions of Logo on Apple and IBM computers, and found several that had turtle shapes for the following degrees: 0 5 15 25 35 45 55 65 75 90, etc. We decided to create turtle icons for every 5 degrees of rotation in the turtle graphics that we created for *HyperCard*. We named Turtle 0 "T0," Turtle 5 "T5," Turtle 90 "T90," and so on.



The turtle icons for the first 90 degrees of rotation looked like this:



Once we created all 72 turtle icons, named T0 through T355, we created a turtle button to put them in. We named the button "Turtle." Be sure that the "Show Name" checkbox is not selected, and that the style of the button is set to transparent, as shown below. (If you have never made a *HyperCard* button, and don't know how to create one, you will need to refer to the instructions in last month's column.) *Note*: to highlight the the important features, we have modified some of the pictures of the dialog boxes so that only the most salient aspects are included in the illustrations.


Button Name: Turtle

Card button ID: 1

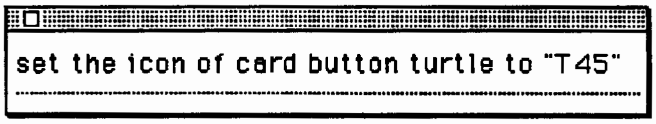
☐ Show name

Style:

[✓ transparent]

opaque

Once you have created the turtle button, you can set the shape that appears in the button to any of the turtle icons made previously. For example, you might type the following in the Message Box of *HyperCard*. (If the message box is not visible, you can make it appear by holding down the Command key – the one to the left of the spacebar with a picture of a cloverleaf – and typing "M.")



```
set the icon of card button turtle to "T45"
```

This will produce the following turtle shape, pointing in a 45 degree direction, in the middle of the turtle button.

Turtle Substitutes

If you have not developed the turtle icons at this point (it took us an evening to make them), you can still complete the remainder of the activities in this column using the compass icon in place of the turtle icons. The turtle button will not be as "turtle like" as traditional versions of Logo if the compass icon is used, but it will still be able to follow traditional Logo commands such as FORWARD and RIGHT. If you are using the compass icon in place of the turtle icons, type this in the message box instead:

```
set the icon of card button turtle to
    "compass"
```

Creating the Right and Left Commands

The commands to turn the turtle will record a heading for the turtle, and then set the turtle icon to the shape nearest that heading. If the turtle heading were 67 degrees, for example, we would want to set the turtle shape to icon "T65", since icons only exist for every five degrees of rotation. To round the heading to the nearest five degrees, first divide by five, round the result, and then multiply by five. To verify that this works, type the following into the Message Box:

```
round (67 / 5) * 5
```

The answer returned in the message box should be "65." If you had typed 68 degrees rather than 67, the algorithm would have rounded the answer up to "70."

If a heading is greater than 360 degrees, it is also necessary to start counting from zero again. For example, if a user types "Right 370", the actual heading should be set to 10 degrees (that is, 370 - 360). This adjustment can be made through use of the "MOD" function, available in both Logo and *HyperCard*. To try out this function, type the following in the Message Box:

```
370 MOD 360
```

The result returned by *HyperCard* should be "10."

These are the basic algorithms that will be necessary to create a "*Turn*" procedure. To develop a command for RIGHT, first record an initial heading for the turtle in the Message Box:

```
put 0 into heading
```

Then go to the Stack editor of the *HyperCard* stack, and enter the following script. (Delete any other scripts which may already be in the editor before entering this one. If you do not know how to enter a script, refer to the preceding two columns.) Remember that in these and other examples the symbol "¬" means that the line is continued, and is generated by holding down the option key as you press the return key.

```
on RIGHT degree
  global heading
  add degree to heading
  if heading < 0 then put ¬
   (heading mod 360) + 360 into heading
  if heading > 355 then put ¬
   heading mod 360 into heading
  set icon of card button turtle to ¬
   "T" & round (heading / 5) * 5
end RIGHT
```

This script will turn the heading of the turtle by the amount specified, and set the turtle icon to the corresponding turtle shape. (*Important note:* If you are using the compass icon rather than the turtle icon, omit the last line of the procedure.)

The procedure RIGHT can be used to create its counterpart LEFT. Degrees are simply multiplied by minus one to turn the turtle in the opposite direction.

```
on LEFT degree
   RIGHT (-1 * degree)
end LEFT
```

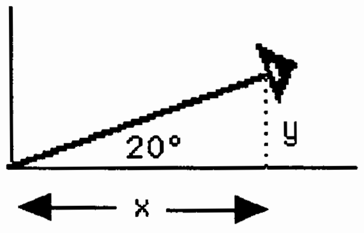
Developing Forward and Back

Once the commands to turn the turtle are created, we are ready to develop commands to move the turtle `FORWARD` and `BACK`. In last month's column we created commands to move the compass button, but commands such as UP and DOWN were limited to movement along right angles. `FORWARD` and `BACK`, in combination with `RIGHT` and `LEFT`, can be used to move the turtle in any direction.

Assume that the heading of the turtle is 70 degrees, and that we would like it to travel forward 100 steps. To put the turtle in the right location, we will need to move it over by X amount, and up by Y amount. Therefore, if we knew the values of X and Y, we would have the coordinates of the new location of the turtle.

Papert designed turtle graphics to provide a "Math Land" in which numbers could be explored in a natural context, but it must be confessed that mathematics still produces anxiety even in some long-time Logo users. If a discussion of numbers provokes undue anxiety, you can skip directly to the procedures for FORWARD and BACK and enter their scripts. However, for those who are interested, we provide an explanation of how the values of X and Y can be determined. Only the simplest arithmetic is required, so you will not need an advanced degree in mathematics to understand the derivation.

The X-value can be calculated by multiplying the distance the turtle has traveled times the cosine of the angle shown below.



This can be expressed algebraically as:
X =
Distance times Cos (Angle)

The angle in this case is the heading of the turtle subtracted from 90 degrees. Since the heading of the turtle is 70 degrees, 90 minus 70 equals an angle of 20 degrees. Therefore the formula needed to calculate X in this instance is:

X = 100 times Cos (20 degrees)

There is only one slight hitch. The cosine function in Logo accepts inputs expressed in degrees, but the cosine function in *HyperCard* requires that measurements be expressed in radians rather than degrees. Fortunately, degrees can be converted to radians by multiplying them by the value of PI divided by 180.

radians = degrees times (PI / 180)

Therefore, the formula to calculate the value of X becomes:

X = 100 times Cos (20 degrees times PI / 180)

A more general formula to calculate the value of X for any turtle heading and distance could be expressed in this way:

X = Distance * Cos (90 - Heading * PI / 180)

This value of X must be added to the current location of the turtle to find the new X position.

The new value of Y is calculated in a way that is very similar to the way in which X is calculated. The value of X is the distance traveled times the *cosine* of the angle, while the value of Y is the distance traveled times the *sine* of the angle.

Y = Distance * Sin (90 - Heading * PI / 180)

This method is used to determine values of X and Y and add them to the current position of the turtle to locate a new position. Once the new coordinates are calculated, the location of the turtle is set to the new position.

```
on FORWARD length
  global heading
  put the loc of card button turtle into
    pos
  put item 1 of pos + length *
    cos ((90 - heading) * PI / 180)
    into x
  put item 2 of pos - length *
    sin ((90 - heading) * PI / 180)
    into y
  put round (x) & "," & round (y)
    into newPos
  set the loc of card button turtle to
    newPos
end FORWARD
```

The procedure FORWARD can be used to create its counterpart BACK. The variable "`length`" is simply multiplied by minus one to move the turtle in the opposite direction.

```
on BACK length
  FORWARD (-1 * length)
end BACK
```

Other Direction and Position Commands

With FORWARD, BACK, LEFT, and RIGHT you can move the turtle anywhere on the *HyperCard* screen just as you can in Logo. There are a few other direction and positioning commands that are handy to have. Sometimes it is convenient to set the turtle heading to an absolute direction. In Logo this is accomplished with the SETH (Set Heading) command. The *HyperCard* equivalent can be written in the following way.

```
on SETH degree
  global heading
  put degree into heading
  if heading < 0 then put ¬
   (heading mod 360) + 360 into heading
  if heading > 355 then ¬
   put heading mod 360 into heading
  set icon of card button turtle to ¬
   "T" & round (heading / 5) * 5
end SETH
```

The *HyperCard* turtle we created does not wrap, but just keeps traveling when it goes off the edge of the screen. The HOME command provides a way to bring the turtle to the center of the screen if it gets lost. Since the *HyperCard* screen is 512 steps across and 342 steps down, X and Y coordinates for the center of the screen would be approximately 256,171.

```
on HOME
  seth 0
  set the loc of card button ¬
   turtle to 256,171
end HOME
```

Because *HyperCard* does not record the value of variables when you exit the program, it will be necessary to initialize the value of the turtle's heading each time the stack is opened. Some versions of Logo have a STARTUP procedure that can be used to automatically initialize setup variables when the program is first started. In *HyperCard* a similar function can be accomplished with an **openStack** script that is placed in the Stack script.

```
on openStack
  global heading
  put 0 into heading
 set icon of card button turtle to "T" & heading
end openStack
```

Entering the openStack script in the Stack editor will ensure that *HyperCard* does not complain that it does not know the value of heading when you first start the program.

Summary

In the previous two months, we showed you how to do the following:

- Create a new stack
- Create *HyperCard* buttons
- Write a HyperTalk procedure
- Use inputs in HyperTalk procedures

In this column we have demonstrated how you can:

- Build a Logo turtle in *HyperCard*
- Create LEFT and RIGHT commands to turn the turtle
- Create FORWARD and BACK commands to move the turtle
- Write an **openStack** script to initialize variables

This turtle can move around the screen just as the Logo turtle does, but it cannot draw yet, because it does not have PENUP and PENDOWN commands. In next month's column, the last of the year, we will give the turtle a pen in *Turtle Graphics for HyperCard.*

Glen Bull is a member of the instructional technology faculty in the Curry School of Education at the University of Virginia. Gina Bull is a programmer analyst for the University of Virginia Department of Computer Science. By day she works in a Unix environment; by night, in a Logo environment.

Glen and Gina Bull
Curry School of Education
Ruffner Hall
University of Virginia
Charlottesville, VA 22903

BITNET addresses:
Glen: GBULL@ VIRGINIA. Gina: GINA@VIRGINIA.

Sources for Icon Editors

Both commercial and shareware icon editors are available. We recommend Icon Factory because it is designed to work directly with *HyperCard.* It is $49.95 direct from the manufacturer.

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 Global News will be introducing both our new Asian correspondent as well as a new Logo book from Europe. The column then looks at several common ways that Logo is introduced into a society.

Greetings

Our new Asian correspondent is Marie Tada. She sends the following greeting to *LX* readers.

I am excited about being an Asian area representative for the *Logo Exchange* and hope to be able to send you interesting updates on what is happening in regards to Logo learning in Japan and in other Asian locations. As this appointment has been very recent and I haven't had time to scout out any other news, I would like to introduce myself and let you know about my connections with Logo. I am presently a computer coordinator at St. Mary's International School in Tokyo and have been living in Japan since 1971. St. Mary's is a boys' school with approximately 1000 students from 70 different countries. It was my good fortune to take a summer course in Logo in Massachusetts about seven years ago. From that time I have taught Logo in summer school sessions and to second grade classes. A few years ago we were fortunate to get a large donation of computers from IBM to set up an elementary school computer lab. Since then all students from Grades K-6 have had weekly lessons in the computer room with LogoWriter at the core of the curriculum. I would like to encourage any *LX* readers with information of interest to Asian Logo users to correspond with me.

Eurologo Proceedings

Logo Exchange readers may well be interested in the proceedings of the Eurologo '89 Conference that was held last September. Eighteen of the papers have been combined in a book entitled *Teaching and Learning in Logo-based Environments*, edited by G. Schuyten and M. Valcke. The articles reflect the wide variety of themes presented and discussed during this international event, including teacher training, Logo research, Logo practice, new Logo versions, hardware extensions, meta-analysis of Logo research and the state of the art and theoretical backgrounds of Logo research and practice.

The contributions can be subdivided into four parts: The first set of texts is of a rather general nature and discusses current theory and practice in Logo environments from three perspectives: first from a constructivist point of view, second from a research point of view and a third based on all the contributions to Eurologo '89. A second set of texts focuses on the pupil in Logo-learning environments. The pupil's learning behavior is researched and described in varying settings. A third set of texts focuses on the teacher, especially on his training status. The next set of contributions concentrates on the computer learning environment; microworlds, hardware extensions and new Logo versions are discussed. Some of the contributions illustrate the relevance of the technical add-ons for enhancing the educational potential of the new learning environments. This book can be summarized as one attempting to answer the questions: What is the real educational potential of Logo-learning environments and how can this potential be realized?

Those interested in obtaining a copy of this book should write: c/o IOS, P.O. Box 2848, Springfield, VA 22152-2848 USA (FAX 703-250-4705) or c/o IOS, Van Diemenstraat 94, 103 CN Amsterdam, Netherlands (FAX 31 20 22 60 55) or c/o IOS, Highway Development Co. Ltd., 1st Golden Bldg., 8-2-9 Ginza, Tokyo-Chouoku, Japan 104 (FAX 81 35 72 86 72)

Logo in Society

I would now like to turn to ways that Logo is introduced into a society, with each way carrying particular implications for how Logo will be accepted by the importing society.

For convenience of discussion, modes of transfer can be identified with three kinds of people who initiate the transfer:

(1) missionary zealots, (2) interested officials in combination with willing helpers, and (3) learners abroad (Michel, 1987, p.128).

Missionary zealots are members of advanced industrial societies who are intent on vigorously disseminating Logo to developing societies. They seek out key members of developing nation's political and educational hierarchies with the intention of convincing those officials to include Logo in their country's instructional system.

Some of these Logo zealots are producers of Logo materials, and in their desire to sell their goods, they oversell their product. Another group of zealots truly have the welfare of Third World peoples at heart. They often represent international organizations (UNESCO, UNICEF), or foreign offices of a government (the British Council, US-AID). In a humanitarian spirit, they advocate the adoption of Logo because they are convinced that it will promote educational progress in developing societies.

Interested officials are members of a developing society who are seeking ways to solve their nation's educational problems, particularly the problem of furnishing widespread, high-quality educational opportunities to their populations at a reasonable cost. The more decision makers in an importing nation know about the educational advantages and disadvantages of Logo, the greater the probability Logo will be adopted in a form that enhances the educational effectiveness of their society.

Learners abroad are either students from developing countries who are in long-term study programs in high-technology societies, or else they are short-term visitors to high-technology nations (e.g., attending conferences). When they return home, they attempt to introduce Logo into their own societies.

A variety of means may be used by these purveyors of Logo to acquaint people of the developing society with the characteristics of the language and philosophy. The principal means are demonstrations, videos, and lectures.

Two characteristics of Logo advocates that influence how Logo will be adopted in a developing society include: (1) the advocates apparent level of expertise regarding both Logo and the recipient society, and (2) how truthful and well-intentioned these enthusiasts appear to be.

The more that the exporters know about the culture and goals of the recipient nation, the more appropriate and efficient the transfer of Logo will be. Ignorance of local conditions can hamper efforts to implement Logo. Suggesting Logo for nationwide implementation in a country where the cost of computers would be enormous, where there is no skilled staff to develop high-quality programs, where there is no electricity in a great many villages, where computers deteriorate rapidly in the humid climate, and where there are no facilities for repairing computers, would be fruitless.

I will be expanding on these and other considerations (political, economic, etc.) for introducing Logo into a society in a paper that will appear in a special issue of *Education* dedicated to Logo. *Education* is in its 111th year of publication and this Logo issue is something to look forward to.

LONG DISTANCE LOGO

Educators–You don't have to go to classes to earn graduate credit–let the classes come to you! Introduction to Logo For Educators, a graduate level independent study course, allows you to learn at your own pace while corresponding with your instructor by mail.

WORK INDIVIDUALLY OR WITH A GROUP

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

NOT JUST ANOTHER CLASS

Dr. Sharon (Burrowes) Yoder, editor of the *Logo Exchange* journal, designed ***Introduction to Logo For Educators*** to provide staff development and leadership training. The four quarter-hour course meets the standards of the College of Education at University of Oregon, and carries graduate credit from the Oregon State System of Higher Education.

ON LOGO VIDEO TAPES

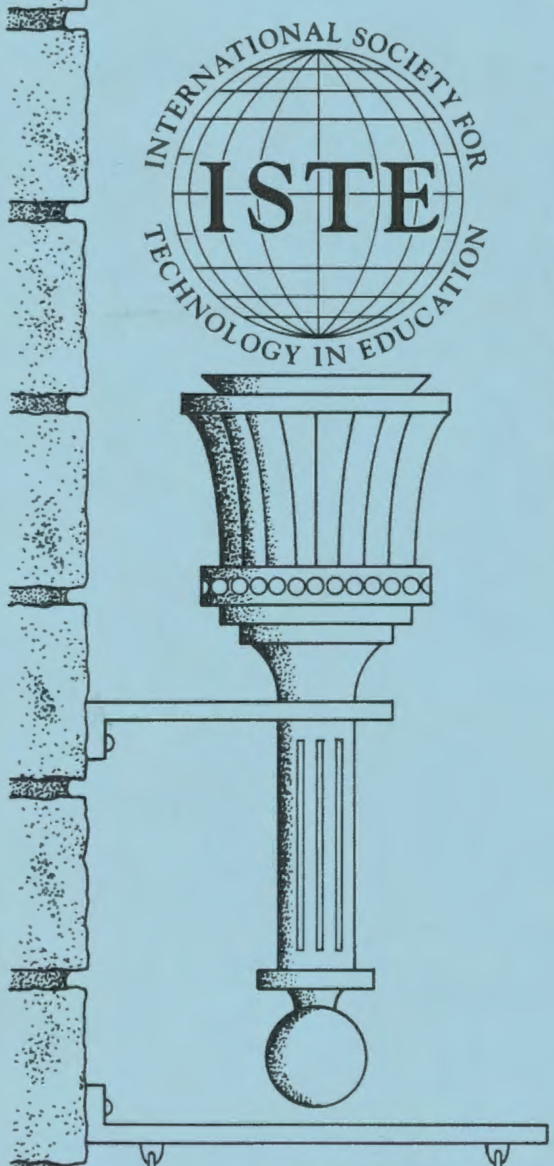
School Districts may acquire a license for the use of the ON LOGO package of 8 half-hour videotapes and 240 pages of supporting print for $599.00. For a one-time fee of $1295.00, the package may be obtained with both tape and print duplicating rights, enabling districts to build libraries at multiple sites.

Group Enrollment. A tuition of $260 per participant is available to institutions that enroll a group of six or more educators. This special price does not include the ON LOGO videotapes. Your group must acquire the tapes or have access to them. Once acquired, the library of tapes and materials may be used with a new groups enrolling for the same reduced fee.

Individual Enrollment. Educators with access to the tapes may enroll indiviidually for $290. Tuition including tape rental is $320. A materials fee fo $60 per enrollee is charged for texts and a packet of articles. Enrollees who already have the texts do not neet to order them.

Tuition Information, Detailed Course Outlines, and Order Blanks can be obtained from:

LONG DISTANCE LEARNING, ISTE, University of Oregon,
1787 Agate St., Eugene, OR 97403-9905.
Phone 503/346-4414



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. $36.00

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. $69.00

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. 503/346-4414