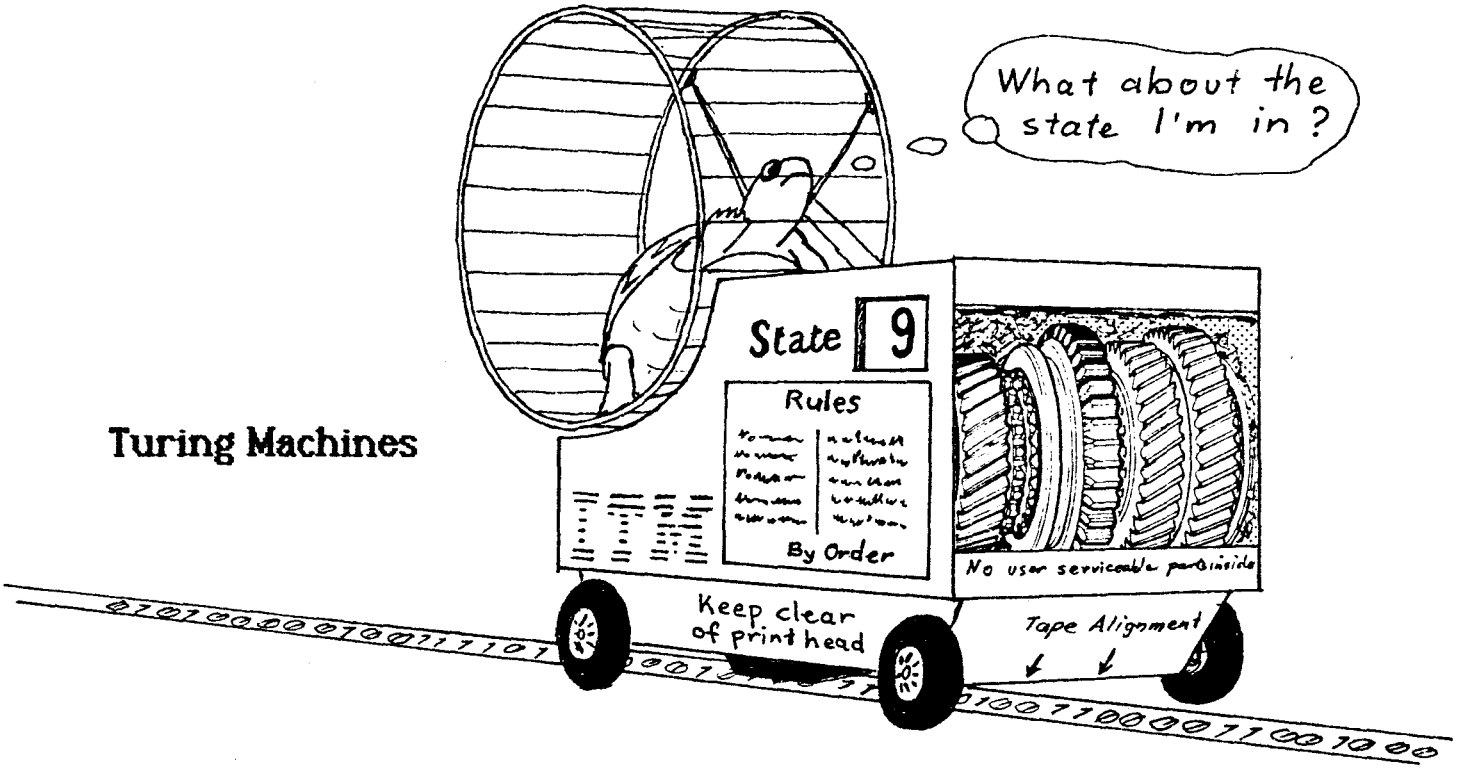
POALL

A Journal for Logo Users



Turing Machines

Volume 2 No 2, April 1987

The unthinkable has happened; *POALL* contains a BASIC program. When you see it you'll understand why.

A certain bureacracy has moved me again. I'm now at Salisbury High School.

The E-Mail age has arrived at *POALL*. Three articles in this issue have arrived, courtesy of the Red Ryder on the Mac, via the Magill VAX. If you have access to that machine, mail your contribution to CARTER, and for Nexus users, the address is PCARTER.

ACEC '87 is fast approaching, and the various committees will soon be editing the papers and preparing them for the proceedings. The Logo papers we know about are listed on p 18. Will be an exciting time.

Theme article for this issue is another in the Computing Science series. Special thanks to the contributors to this issue.

Peter J. Carter

POTS

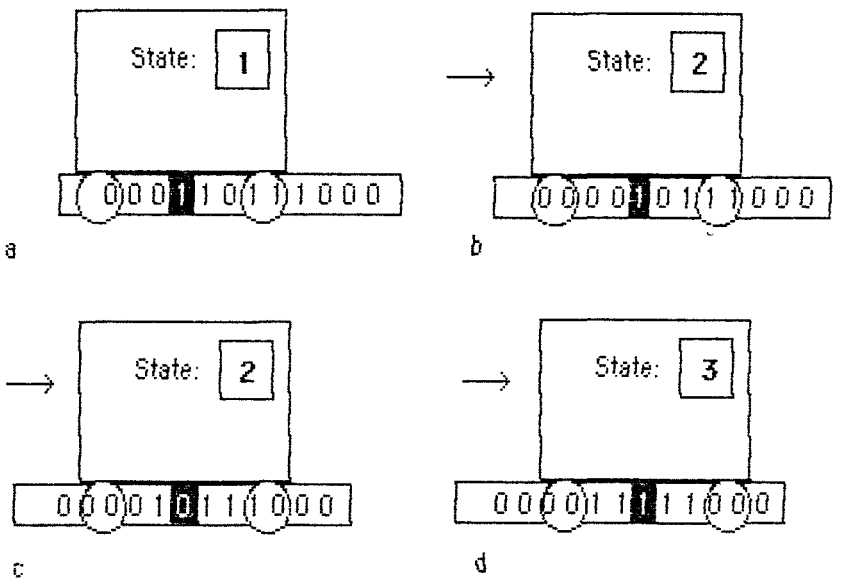
Turing Machines:

Alan Turing (1912..1954) was a brilliant, but tragic, figure in the history of computing. During the war he was a key member of the team at Bletchley Park that decoded German communications, securely encrypted, so the German High Command believed, by the Enigma machines. The full extent of the work at Bletchley Park has never been revealed, but it was to lead to the development of Colossus, arguably the world's first, if specialised, electronic computer*. Had the Enigma codes not been broken the world would almost certainly be a very different place today.

In 1936 Turing published a paper, 'On Computable Numbers...' That paper defined the limits of computability, what computers could and could not do. He made use of a theoretical machine, the 'Turing machine', which he did not propose to build, but used as an example of logical reasoning. It stands as a model of all modern computers.

A Turing machine consists of a device to read from and print on a paper tape marked with 1s and 0s. It can move along the tape, reading and printing one character at a time according to the rules incorporated into its mechanism. The rules describe the 'states' the machine is to adopt as it moves to and fro along its tape.

As John Hopcroft says, 'The best way to understand how a Turing machine works is to try to build one.' Rather than assemble cogs and things, all we need to do is specify the rules needed to perform some action. For example, let's make a machine to add two numbers, 2 and 3, which will be represented on the tape by 000110111000. We'll start the machine with its read/print head over the leftmost 1, and in what we'll call state 1. The easiest way to do it is for the machine to move the 11 string against the 111 across the 0, moving only one step at a time. To do that we'll let the machine move to the right until it comes to the first 1 in 11 (a in the diagram below). It changes the 1 into a 0, changes itself to state 2 and moves to the right. In state 2 it takes no notice of any 1s it finds (b) but moves right. When it finds a 0 it changes it to a 1, enters stage 3 and stops (c and d). The final result on the tape is 000011111000.



* Contrary to what some books may suggest, Colossus was not used on Enigma codes but on messages encoded by Geheimschreiber, a modified teleprinter with 10 encoding rotors. Enigma lives on in the *crypt* command of the UNIX operating system. As the manual states: 'Crypt implements a one-rotor machine designed on the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work is likely to be large.' It was. Some 10 000 people worked on the decoding task for the six years of the war.

The rules built into a Turing machine must either cause it to stop or specify three things: what it must print on the tape, the state it must assume, and whether it is to move to left or right. The rules for our first machine are these:

State	Symbol	Print	New State	Movement
1	0	0	1	R
1	1	0	2	R
2	0	1	3	R
2	1	1	2	R
3	0	Stop		
3	1	Stop		

Rules like that can be expressed in Logo as well as a table like that, but before we try we'll need to decide a couple of things. Our tape will be in the form of a word, a free variable. The machine's position on the tape will also be a free variable, and the two will be free because the machine 'knows' nothing of the rest of the tape or its position. The only things represented in it are its state and the rules. We will need some way of reading the tape, printing on it and moving the machine. On the screen we'll represent the tape by simply printing it and the machine by printing its state above the 'tape'. That leads to these procedures (Note that this is not a real, but an idealised Logo. You *will* need to make changes, especially to the IF..THEN..ELSE line in Move):

```
TO ReadTape
OUTPUT ITEM :position :tape
END

TO Put :symbol
SETCURSOR SE :position * 2  6
TYPE :symbol
MAKE "tape Replace :position :symbol :tape
END

TO Replace :position :symbol :tape
IF EMPTY? :tape [OUTPUT "]
IF 1 = :position [OUTPUT WORD :symbol BUTFIRST :tape]
OUTPUT WORD FIRST :tape Replace :position - 1 :symbol BUTFIRST :tape
END

TO Move :direction :state
SETCURSOR SE :position * 2  5
REPEAT 2 [TYPE CHAR 32]
IF :direction = "R
        [MAKE "position :position + 1]
        [MAKE "position :position - 1]
SETCURSOR SE :position * 2  5
TYPE :state
END
```

Now, let's pseudocode the machine itself:

read the tape
if the state is 1 and the symbol is 0, then print 0, assume state 1 and move right
if the state is 1 and the symbol is 1, then print 0, assume state 2 and move right
if the state is 2 and the symbol is 0, then print 1, assume state 3 and stop
if the state is 2 and the symbol is 1, then print 1, assume state 2 and move right

As you may gather from that, the Logo could be lines of **IF AND ... THEN**s, but if we do that we will eventually have very large, cumbersome procedures. Instead, we'll go this way...

```
TO Machine :state :rules
LOCAL [symbol instructions]
MAKE "symbol ReadTape
MAKE "instructions LookUp SENTENCE :state :symbol :rules
IF EMPTY? :instructions [Halt STOP]
Put FIRST :instructions
Move FIRST BUTFIRST :instructions LAST :instructions
Machine LAST :instructions :rules
END

TO LookUp :conditions :rules
IF :conditions = FIRST FIRST :rules [OUTPUT LAST FIRST :rules
LookUp :conditions BUTFIRST :rules
END
```

...with the rules as a list:

```
MAKE "rules1 [[[1 0][0 R 1]][[1 1][0 R 2]][[2 0][1 R 3]][[2 1][1 R 2]][3 1][]]]
```

The first sublist in each sublist, eg. [2 0], is the state and symbol read, the second, [1 R 3] is the symbol to be printed, the direction to move, and the new state. A [] is the signal to halt.

To make life easier we could do with a couple more procedures, and a ▵ represents a space to be printed:

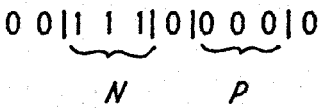
```
TO Halt
SETCURSOR [0 10]
PRINT "Finished
END

TO Setup
LOCAL "state
CLEARTEXT
PRINT [Turing Machine]
SETCURSOR [0 15]
TYPE [Enter the tape:\▵]
MAKE "tape READWORD
PRINT "
TYPE [Enter the starting position:\▵]
MAKE "position READWORD
PRINT "
TYPE [Enter the initial machine state:\▵]
MAKE "state READWORD
PRINT "
TYPE [Which machine?\▵]
Start READWORD :state
END
```

```
TO Start :machine :state
SETCURSOR SE :position * 2  5
PRINT :state
SETCURSOR [2 6]
PrintTape :tape
Machine :state THING WORD "rules :machine
END
```

```
TO PrintTape :tape
IF EMPTY? :tape [STOP]
TYPE WORD FIRST :tape CHAR 32
PrintTape BUTFIRST :tape
END
```

Try it first with the values given in the example earlier, 000110111000 for the tape, 4 for the position and 1 for the state, then with other tapes. Try starting on one of the leading 0s and see if that makes any difference.

Adding two numbers like that may not be the most exciting so let's try something more complex. This time we'll start with a tape like this:



and copy the 111 across to finish with 00111011110. The 111 on the left we'll consider as being in a register named *N*, the result will be in register *P*, and the 0 in the middle is the break between them. Here are the rules:

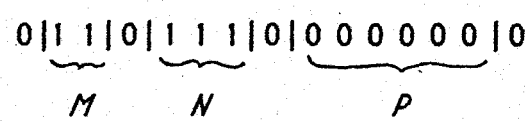
State	Symbol	Print	New State	Movement	Comment
1	0	0	1	R	Move to left end of *N*
1	1	0	2	R	Mark left end of *N* with 0
2	0	0	3	R	Move across break
2	1	1	2	R	Move across *N* to break
3	0	1	4	L	Copy 1 at right end of *P*
3	1	1	3	R	Move past 1s in *P*
4	0	0	5	L	Move left across break
4	1	1	4	L	Move left across *P*
5	0	1	8	L	0 to left of break shows copying complete
5	1	1	6	L	Move left from break into *N*, change state
6	0	1	7	R	Begin move of 0 marker to right
6	1	1	6	L	Move left across *N*
7	0				Doesn't happen in this machine
7	1	0	2	R	Complete move of 0 to right
8	0	0			Finished
8	1	1	8	L	Move left to end of *N*

Let's put that into Logo (Use EDN. You'll probably find it easier to type it like this and then remove the <Return>s):

```
MAKE "rules2 [[[1 0][0 R 1]] [[1 1][0 R 2]]
      [[2 0][0 R 3]] [[2 1][1 R 2]]
      [[3 0][1 L 4]] [[3 1][1 R 3]]
      [[4 0][0 L 5]] [[4 1][1 L 4]]
      [[5 0][1 L 8]] [[5 1][1 L 6]]
      [[6 0][1 L 6]] [[6 1][1 L 6]]
      [[7 1][0 R 2]] [[8 1][1 L 8]]
      [[8 0][]]]
```

Try it with tapes like the one earlier and watch the machine shuttle forth and back. It takes 30 moves with that tape, starting at position 1 in state 1.

Let's now build on that machine to produce one that multiplies two numbers on tapes like this:



The 111 is in *N* as before, the 11 is in *M* and the answer will appear in *P*. A couple of rules are slightly changed, the rest are simply added to the last set, and no, there isn't a state 9. The machine will work by copying, which is why it incorporates the copying machine, and you can begin at position 1 in state 10:

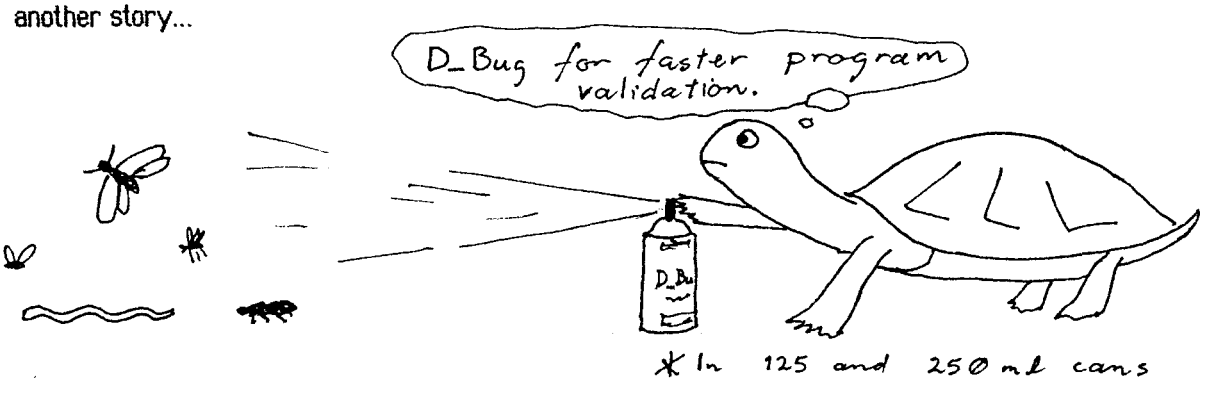
State	Symbol	Print	New State	Movement	Comment
8	0	0	12	L	Move left between *M* and *N*
8	1	1	8	L	Move to left end of *N*
10	0	0	10	R	Move to left end of *M*
10	1	0	11	R	Mark left of *M* with 0
11	0	0	1	R	Begin copy
11	1	1	11	R	Move across *M* to break
12	0	1	15	L	0 found to left of break between *M* and *N*, product in *P*
12	1	1	13	L	Move into *M*, must change state
13	0	1	14	R	Begin move of 0 marker in *M*
13	1	1	13	R	Contine across *M*
14	0				Doesn't occur
14	1	0	11	R	Move 0 in *M*
15	0				Finished
15	1	1	15	L	Move to left end of *M*

MAKE "rules3 [[[1 0][0 R 1]] etc...

You may think all that is a very complicated set of rules just to multiply two numbers, and you're probably right. But that is how *any* computer works, one tiny binary step at a time, and we can define Turing machines to do anything that can be done by computers. The Turing machine is a model of all computers; a modern computer is a 'universal' Turing machine, with its memory taking the place of the tape and a program the rules.

The point of Turing's paper was that he was able to prove that any mathematical function was computable if it could be computed by some Turing machine, in other words by some specified set of logical steps, an algorithm. To understand why that was important you need to know about the work of Russell and Whitehead, Gödel, Church and others in the early part of the century. That's beyond us here, but it lies at the base of much of what we do in computing. Like George Boole, Alan Turing gave us a logical foundation.

You will find Turing machines mentioned in many places, one being Chapter 2 of Weizenbaum's *Computer Power and Human Reason*, but unfortunately the example he gives doesn't work. The examples here came from 'Turing Machines' by J. E. Hopcroft in *Scientific American*, May 1984, which explains the mathematics in depth. J. David Bolter describes them in *Turing's Man*, along with many of the implications of computing. The best description of Enigma and the work of the British Code and Cypher School is in B. Johnson's *The Secret War*. The biography of Turing, *Alan Turing: The Enigma* by A. Hodges covers the whole field in detail. You will almost certainly come across reference to the 'Turing Test' in your reading, but that's another story...



Resources:

The second of Brian Harvey's *Computer Science Logo Style* series is now available, subtitled *Projects, Styles, and Techniques*. It doesn't really need reviewing, simply recommending to everyone using Logo in secondary or higher education or interested in learning more about and with Logo.

Layout and style follow those of the first book, and as the title suggests, this volume is concerned with programming styles and planning methods, explained within the contexts of the ten projects in the book. There are five sections, dealing with cryptography, games, mathematics, utilities and pattern matching. The games described are solitaire and tic-tac-toe, the maths includes Fourier analysis (for musicians) and the utilities include an iteration compiler. This one takes a program written with mapping and modifies it to tail recursive form. To illustrate pattern matching there is a version of DOCTOR, but unlike the simplistic one in Abelson's *Apple Logo* this one uses a two tier structure like Weizenbaum's original. Complete listings are in the book, but the programs are available on disk for users of Apple Logo //, IBM Logo and Macintosh Logo.

Volume 3 should be available some time this year, but Volume 2 has plenty to keep you busy until it arrives. Mentioned by Harvey is *LogoWorks: Challenging Programs in Logo* by Cynthia Solomon, Margaret Minsky and Brian Harvey (McGraw-Hill, 1985). Australian price is quoted as $67, and we haven't taken the plunge yet (Harvey's book is listed at $48.50). On order is Cynthia Solomon's book *Computer Environments for Children: A reflection on Theories of Learning and Education* (MIT Press, 1986). Will be reviewed when it arrives.

Logo rates only a minor comment in *Programmers at Work*, a book of interviews edited by Susan Lammers and published by Microsoft Press. Nineteen programmers describe their styles, their personalities and something of their methods.

Gary Kildall has this to say about problem solving:

'Part of the programming process is general problem solving. How do you solve a problem that's complex, whether it's designing a computer program or constructing a building? You start at the point where you think it's too hard to solve, and then you break it down into smaller pieces. That's what I try to teach.' (p 58)

Bill Gates, who admits to having gone 'to the garbage cans at the Computer Science centre and I fished out listings of their operating system' to find good examples of programming, is also strong on the thinking process:

'Before I sit down to code something, most of the instructions have already run through my head. It's not all laid out perfectly, and I do find myself making changes, but all the good ideas have occurred to me before I actually write the program. And if there is a bug in the thing, I feel pretty bad, because if there's one bug, it says your mental simulation is imperfect. And once your mental simulation is imperfect, there might be thousands of bugs in the program. I really hate it when I watch some people program and I don't see them thinking.' (p 77)

That last sentence is the important one. Logo can be successfully doodled with, without everything having run through one's head beforehand. But the thinking process is vital. Perhaps the most fortright words come from Butler Lampson:

'To hell with computer literacy. It's absolutely ridiculous. Study mathematics. Learn to think. Read. Write. These things are of more enduring value. Learn how to prove theorems: A lot of evidence has accumulated over the centuries that suggests this skill is transferable to may other things. To study only BASIC programming is absurd.' (p 38)

Logo is as good a medium for learning mathematics and proving theorems as anything. A very interesting book.

Another non-Logo book is *Microchip* by T. R. Reid and published by Pan. Reid writes for the Washington Post and his book reads well, even if a couple of minor points are doubtful. It's a history of the developments, including the litigations and technical details, of integrated circuitry. There are some good ideas on the problem solving process, especially from Jack Kilby:

'...Somewhat simplified the method involves two levels of concentrated thought.

At first, the problem solver has to look things over with a wide-angle lens, hunting down every fact that might conceivably be related to some kind of solution. This involves extensive reading, including the obvious technical literature but also a broad range of other publications... "That's all right." Kilby says, "You read everything-that's part of the job. You accumulate all this trivia, and you hope that someday maybe a millionth of it will be useful." For recreation, Kilby says, "I read trash."

The next step in Kilby's system requires switching to an extremely narrow focus, thinking strictly about the problem and tuning out the rest of the world. This requires, first of all, an accurate definition of the problem. "The definition of the problem becomes a major part of the innovation," Kilby has written. "A lot of solutions fail," he says, "because they're solving the wrong problem, and nobody realises that until the patent is filed and they've built the thing." It is also necessary to develop a clear understanding of the natural constraints surrounding the problem; the heart of the inventor's job is finding a way to slip past the roadblocks erected by nature...

In this concentrated, single-minded focus on the question at hand, the problem solver must also tune out all the obvious solutions. This is a key principle, important to emphasize because it is somewhat counter-intuitive. The mind tends to jump to the answer that is immediately evident. In fact, this answer is probably wrong. If the problem is of any importance, all the obvious solutions have been tried already...Some of history's most important innovations, he says, were so nonobvious as to violate the scientific rules of the day. "You only arrived at the invention when somebody developed a method that everyone else had already decided was obviously wrong."' (pp 56..57)

Traditionally, Logo has always had something of a mathematical bias. Seymour Papert on *LogoWriter*:

'...Many teachers who feel more comfortable dealing with domains that focus on verbal or dramatic content rather than on mathematical concepts will now have access to the Logo culture. My second remark anticipates the inevitable question: 'But will this skill of making LogoWriter presentations transfer to increased writing skills in general?' My answer is simple. If you see transfer as an automatic process that needs no encouragement from you, it may or may not. But I am convinced that your imagination as a teacher will show you how to use LogoWriter programming as a transition to pure writing.' (Snarfed from *Softimes*, who snarfed it from *Classroom Computer Learning*.)

If you're working in primary school with Apple or IBM, do have a look at LogoWriter. A Commodore 64 version is apparently on the way. It's an expensive package, but includes site licence, lots of excellent teaching ideas and materials, and even 'take home' licences.

For the musically inclined, Terrapin recently released Terrapin Music Logo for the Apple. It can be used to compose music in six voices with normal Logo procedures through an ALF synthesizer with amplifier, and comes with comprehensive documentation. Terrapin have also released the *LogoWorks™* support materials, with the first being *LogoWorks: Lessons in Logo* and *The Logo Project Book: Exploring Words and Lists*. More are on the way.

TLC-Logo Beta 1.4 for the Commodore Amiga

A Logo is on the way for the third of the 68000 micros, the Commodore Amiga. The Mac has had LCSI/Microsoft Logo and ExperLogo for some time, and the Atari ST machines have DR Logo supplied with them.

TLC-Logo is, well, different, being based on a rather different model of what Logo should be, and it would be as well to consider Logo's ancestry for a moment. Logo derives from LISP, *the* language of artificial intelligence for nigh on 30 years, but its designers changed a few things along the way, mainly to remove the parentheses. Where LISP would say **(Thingo 1 2 3)** Logo has **Thingo 1 2 3** . Likewise, **(Thingo (Wotsit 5 6) 7 8)** becomes **Thingo Wotsit 5 6 7 8**. The Logo looks easier without (the ()) but the problem comes with expressions like **(Thingo 1 2 size)** which cannot be represented as **Thingo 1 2 size**. That word size is not the name of a procedure, it is a variable, so to distinguish it, : (dots), as in :size. (Dots has a meaning, it's not just a convenient label. :size is shorthand for THING "size, in other words, 'return the value bound to the word size'.)

So far so good, but in the opinion of John Allen, president of The LISP Company that makes TLC-Logo, there were some liberties taken. LISP is functional, that is, every module of code returns a value, even if it is only nil or t. The usual Logo is procedural, and for a value to be returned it must be OUTPUT. To Allen that is a disadvantage, and TLC-Logo's major difference from the Logos from LCSI, Terrapin etc. is that it is functional. Something of what this means in practice will be seen from sample procedures later. Another significant difference is that every function is a perfectly ordinary list; the TEXT operation of conventional Logos is unnecessary.

TLC-Logo Preliminary Version Beta 1.4 for the Amiga came with 12 badly photocopied pages of documentation, simply a list of the primitives. No instructions on operating the system, just the primitives, many of which behave somewhat unexpectedly, and not always consistently with what is in the book by John Allen and his colleagues, *Thinking About [TLC] Logo* Understandably the team at Angle Park wasn't very impressed, especially as noone could work out how to define procedures from the editor.

The list of primitives is interesting however, including ACT to activate multiprocessing, CASE, which works like the Pascal one even if the syntax is different, FOR, an extended REPEAT, MAP and MAPQ. There is COND, LISP's equivalent of IF..THEN..ELSE, and DO, something like Logo's RUN. Many predicates take the form ISEMPTY, ISFUN, ISMINUS, ISTHING etc. A potentially useful list mutating primitive is SET. TLC-Logo can HATCH multiple Turtles, the default one being 'STUDS The original, inevitable, turtle'*, but addressing them is not a matter of TELLing them as in other versions. The default shape is an odd oval, actually stretched {}, but shapes can be redefined.

First attempts brought forth some interesting error messages, including this one:

Error: If doesn't like nothing near polyspi

Just the thing to start an English lesson. Another, after a real crash:

Guru Meditation #00000005.0000A708

Now you know what system programmers do in their spare time.

* There is a play on words here. Studs is named after Studs Turkel the sociologist.

Eventually I got PolySpi to work (TLC-Logo is case insensitive, but defaults to lower case):

```
to polyspi :size :angle :inc
 :size > 200
 iffalse [fd :size rt :angle polyspi :size + :inc :angle :inc]
end
```

Remember that every piece of code returns a value, so the line **:size > 200** is equivalent to **TEST :size > 200** in the Logo you're used to. TLC-Logo does **polyspi 1 123 1** in a little over 3 seconds, or about 8.5 with Studs visible. MacLogo takes over 14 seconds.

There are some demonstrations on the disk. One of them draws a good resemblance of M. C. Escher's 1964 print 'Square Limit'. Another has a number of traditional fractal graphics, including:

```
to hilbert :size :level @fn @gn
 :level = 0
 iffalse [fn 90 hilbert :size :level - 1 @gn @fn fd :size gn 90 hilbert :size
:level - 1 @fn @gn fd :size hilbert :size :level - 1 @fn @gn gn 90 fd :size
hilbert :size :level - 1 @gn @fn fn 90]
end
```

Two things about that. The good thing is that functions can be easily passed as arguments (the @fn business). The bad thing is that the formatting of functions is absolutely abominable, and nothing like what is shown in *Thinking About [TLC] Logo*, or possible in any LISP system. Without knowing how the editor works it's impossible even to enter a function like that; when the cursor reaches the edge of the screen that's it. Something that really does need attention before final release.

I was determined to get a list processing function to work, one I often use in various forms:

```
TO Remove :item :object
IF EMPTY? :object [OUTPUT []]
IF :item = FIRST :object [OUTPUT Remove :item BUTFIRST :object]
OUTPUT SE FIRST :object Remove :item BUTFIRST :object
END
```

That's best done with COND in TLC-Logo, but I had to abbreviate to put it all on one line:

```
to r :i :o
 cond [[isempty :o []][:i = first :o r :i bf :o]["true fput first :o bf :o]]
end
```

Yuk! This would have been more like it:

```
(de remove (item object)
        (cond ((null object) nil)
              ((eq item (car object)) (remove item (cdr object)))
              (t (cons (car object) (remove item (cdr object))))))
```

Besides being fast, TLC-Logo has more stack space than anything else around. This function stopped with the very descriptive message **Error: error** with :number at 4095:

```
to thingo :number
printnl :number
thingo :number + 1
printnl "#
end
```

The figure for MacLogo (default stack on 512k machine) is 1389, and for the disastrous Atari ST Logo, 217. I wanted to try to find the maximum length of list that Remove, alias r, could handle, but couldn't work out how to cons up long lists. SE and WORD do rather different things from standard Logo versions.

The Amiga's windows and mouse look superficially like the Mac's. Perhaps as a Mac user I've been spoiled, but I found the Amiga system somewhat cumbersome. The colour in some of the demonstrations was really good though.

A school can buy an Amiga for $1495 (512k, single drive, colour monitor), say $2000 for machine and software. Will TLC-Logo be worth having with it? I think it depends. For a primary school I have my doubts, and for high schools too, if recent experience is any guide. One of the problems with Logo has always been that it is harder for teachers than many other items of software. Not every teacher has had the opportunity to spend hours and hours playing about with Logo or to be coached by someone with experience. TLC-Logo is so different from other Logos and the materials in books that most teachers are likely to have so many problems they will give it away. That's unfortunate, because TLC-Logo isn't difficult, in fact some things are easier, but there are all too few support materials, only *Thinking about [TLC] Logo* at present. Some will find its style aggravating, but the book is an excellent discourse on the philosophy of Logo in general and TLC-Logo in particular. I wouldn't be surprised if it forms part of the system's documentation.

Despite, perhaps because of, its differences, TLC-Logo is a very powerful system, more advanced in several aspects than the usual Logo, with vectors, streams, environments, multiprocessing and the like. They are features needed for university level teaching and AI research. This Logo's place may well be in tertiary institutions where it can be supported by people with LISP experience who understand and appreciate its power.

Without denigrating TLC-Logo in any way, for most Amiga users I think a conventional Logo would be better. Mac users have a choice and so do users of MS-DOS machines. Perhaps Commodore might commission an Amiga Logo from Terrapin, who make Logo for the C 64.

My thanks to Ralph Leonard and the team at Angle Park for the opportunity to try it.

The book:

Allen, J. *et al* *Thinking About [TLC] Logo* Holt, Rinehart and Winston, 1984

(Reviewed in the first issue of *POALL*)

The manufacturers:

The LISP Company 430 Monterey Ave. #4 Los Gatos California 95030

(As the name suggests, they supply (highly regarded) LISP systems.)

Thinking Logo by Peter J Carter

Reviewed and strongly recommended by Hartley Hyde

Thinking Logo is a book which follows more closely than most the Logo path to educational growth which was first visualised by Papert (*Mindstorms*, 1980). While reference is made to useful techniques, the book is one which encourages the reader to think, rather than learn to program.

During the last five years I have collected a scrap book of Logo teaching ideas; photostats of magazine articles and serviettes borrowed from dinner tables at which the conversation had turned lightly to a discussion of Logo procedures. After reading Carter's new book, I now face a major spring cleaning task, whereby most of this material can be discarded. Thinking Logo provides an encyclopaedic coverage of the topic. Whereas most text books agonise through the traditional turtle graphics chapters and then give passing reference to just a few of the more interesting aspects of the language, Carter has not only collected all of these concepts into one volume, but he has also woven them into a clear, logical educational treatise.

Carter is currently listed by the prestigious user group magazine *Call A.P.P.L.E.* as one of only two Logo consultants in the world. He is a child of the Logo universe: but his expertise goes largely untapped in our community. The wisdom expressed in St. Luke 4: 24 continues through the ages. Carter's contribution is seriously underestimated by South Australians. We should compare the high quality of Peter's articles in this magazine with those of more glossy overseas publications.

Yes, criticisms will be made. Because the book is encyclopaedic, teachers who wish to promote a particular narrow view of education through Logo, will be frustrated by the variety of different paths which Carter pursues. For example, those who might examine the book as a Computer Studies text will find much useful information, but they will also find many sound educational practices which fall outside the narrow view imposed by the intellectual disipline of that subject. In some ways, Carter has used this book to preach a message guiding us back toward the Papert concept, away from those who would limit our use of the language to a particular and narrow purpose.

Finally, readers of this magazine will understand, but others should be warned, that the Carter concept of wit is unique: it should be aquired slowly or it may hinder communication. Who else would reference a book in its own Bibliography as:

> An amorphous and idiosyncratic book for students to learn programming through Logo, with a self-referential bibliographical entry.

Fractals - Logo style

David D. Curtis, SACAE Magill

The attached BASIC listing was published in *Online* [1] recently. It surely is a case of using an inappropriate tool to do a job. The program attempts to simulate recursion in a language which was not developed to support it and which does not support local variables. It does this by creating and manipulating string variables. How good a simulation it is is not being debated here, although it is an interesting programming exercise. It is however instructive to compare the listing with a pair of Logo procedures that do the same job.

Inspection of the code presented in the two languages reveals that the Logo procedures are much more compact and are I believe more readable. Readability of code is very important, especially in education, where programs are a means of communicating programming ideas with students and peers as well as communicating commands to a machine.

The BASIC listing is written for a Microbee using MicroWorld BASIC which is one of the versions of that language supplied with the machine. Variable names are a little different from those used in other dialects in that integers are represented by single letter variable names, reals are represented by a single letter followed by a number from 0 to 7, while strings are represented as are reals except for the appended $ symbol. Integers and reals may not generally be freely mixed, hence the use of the FLT and INT functions. The syntax of string functions is a little different from many other dialects. A0$(;M,N) is equivalent to MID$(A0$,M,N). Constants 511 and 255 reflect the 512 by 256 dot resolution on the graphics screen.

The variables used in the Logo procedures (:S, :L and :N) represent the number of sides of the base figure, the length of each side of the base figure and the number of levels desired for the figure.

Logo Procedures:

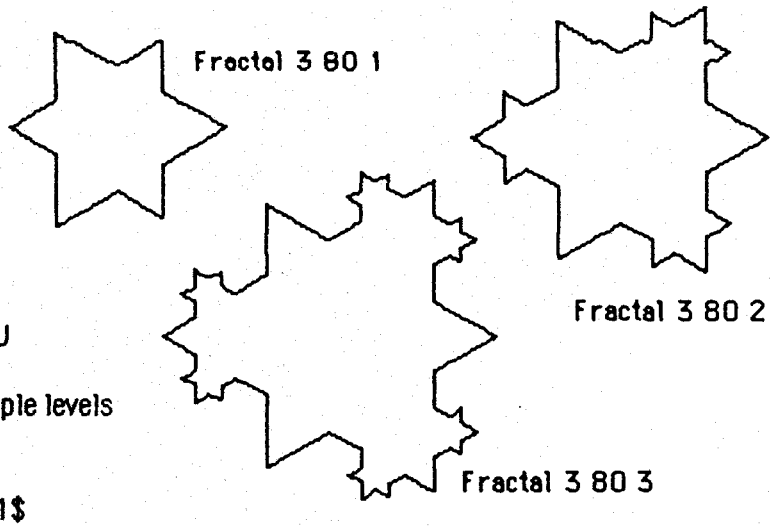
```
TO FRACTAL :S :L :N
IF :N = 0 [REPEAT :S [FD :L RT 360/:S]]
      [REPEAT :S [FD :L/:S LT ( 180-360/:S) FRAC :S :L/2 :N-1
                              FD :L/:S RT 360/:S]]
END

TO FRAC :S :L :N
IF :N = 0 [REPEAT :S-1 [FD :L/2 RT 360/:S] LT 180]
      [REPEAT :S-1 [FD :L/:S LT ( 180-360/:S) FRAC :S :L/2 :N-1
                           FD :L/:S RT 360/:S] LT 180]
END
```

BASIC listing:

```
00100 REM Fractal program for the Microbee
00110 REM Examples 3 N 1, 3 N 2, 3 N 3, 3 Y 3, 3 Y 4
00120 REM 4 Y 2, 4 Y 3, 5 Y 1
00130 B0$="":A1$="":B1$=""
00140 C4=3.1415927/180:M6=511:M7=255
00150 INPUT"Number of sides? ";S
00160 C6=M6*.7:C7=M7*.7:0=INT(C6):H=INT(M7-C7)+1
00170 INPUT"Inverse? (Yes/No) ";I0$:I0$=I0$(;1,1)
```

.....continued

```
00180 REM Angles to turn left and right
00190 R0=-360/FLT(S):L0=R0+180
00200 IFI0$="Y" OR I0$="y" THEN LET C0$="A":W0=R0:R0=-L0:L0=-W0
      ELSE LET C0$="R"
00210 A0$=C0$
00220 IFI0$<>"Y" AND I0$<>"y" THEN 290
00230 A1$="R":T=S-2
00240 FOR N=1 TO T
00250  A1$=A1$+"L"
00260 NEXT N
00270 A1$=A1$+"R"
00280 GOTO 340
00290 A1$="L":T=S-2
00300 FOR N=1 TO T
00310  A1$=A1$+"R"
00320 NEXT N
00330 A1$=A1$+"L"
00340 INPUT"Number of levels? ";J
00350 IFJ=1 THEN 450
00360 REM Expand string for multiple levels
00370 FOR N=2 TO J :K=LEN(A0$)
00380  FOR I=1 TO K
00390   B0$=B0$+A0$(;1,1)+A1$
00400  NEXT I
00410  A0$=B0$:B0$=""
00420 NEXT N
00430 REM Scale size of diagram to fit screen
00440 REM The factor ( 1.49) has to be reduced for more complex figures
00450 L1=M6*1.49/(FLT(S)*3^FLT(J))
00460 HIRES2
00470 FOR Z=1 TO S
00480  FOR N=1 TO LEN(A0$)
00490   B1$=A0$(;N,N)+A1$
00500   FOR I=1 TO LEN(B1$)
00510    B0$=B1$(;I,I)
00520    IF B0$="A" THEN LET V0=V0+W0:GOTO 540
00530    IF B0$="L" THEN LET V0=V0+L0 ELSE LET V0=V0+R0
00540    A7=V0*C4
00550    X0=X0+L1*COS(A7):Y0=Y0+L1*SIN(A7)
00560    X=INT(X0):Y=INT(Y0):D=X+INT(C6):E=INT(M7-Y0*.58-C7)
00570    PLOT G,H TO D,E
00580    G=D:H=E
00590   NEXT I
00600  NEXT N
00610 NEXT Z
```



Reference:

MacLachlan,K.R. 'Fractal'. *Online 28* p38. October, 1986

Editor's Comment:

I can't speak for the BASIC, but the Logo version does work, as shown by the samples. But are they real fractals? Compare them with the sample SnowFlakes elsewhere in this issue. Why do people still advocate BASIC in education?

Scissors

More adaptations from the pages of *Turtle Geometry*

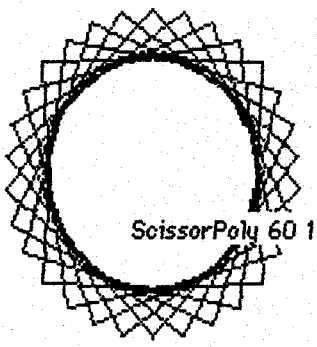
The usual Poly procedures, PolySpi, DuoPoly, MultiPoly, and all the rest, are based on straight lines. That need not be. We can replace each line segment with a scissors:

```
TO Scissor :distance :angle
RIGHT :angle
FORWARD :distance
LEFT 2 * :angle
FORWARD :distance
RIGHT :angle
END
```

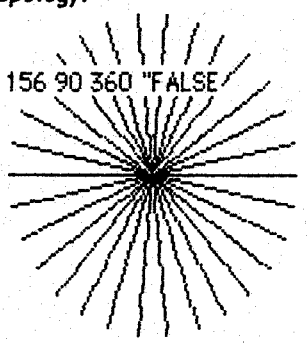
Now we can define a ScissorPoly, complete with stop rule:

```
TO ScissorPoly :distance :angle :phase :totalTurn :doneOne
IF AND :doneOne 0 = REMAINDER :totalTurn 360 [STOP]
Scissor :distance :phase
LEFT :angle
ScissorPoly :distance :angle :phase :totalTurn + :angle "TRUE
END
```

As you try it, think about how that stop rule works, and watch how the shape changes as the phase changes. Think also about the symmetry and topology.



ScissorPoly 60 132 20 360 "FALSE

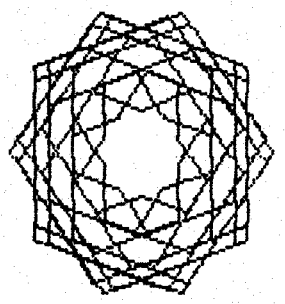


ScissorPoly 60 156 90 360 "FALSE

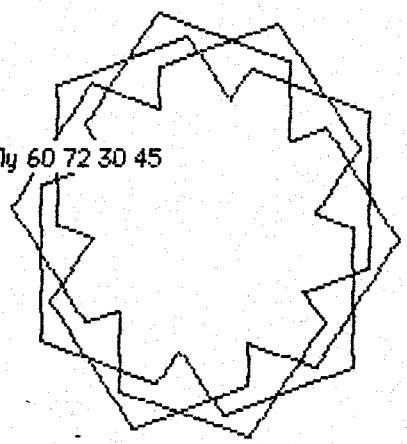
We could, instead of bending each segment, shrink it:

```
TO ShrinkPoly :distance :angle :localPhase :localPhaseChange
FORWARD :distance * COS :localPhase
LEFT :angle
ShrinkPoly :distance :angle :localPhase + :localPhaseChange :localPhaseChange
END
```

Segments disappear when :localPhase = 90. Study how these shapes change with changing :localPhaseChange, and invent a stop rule.



ShrinkPoly 60 72 30 20



ShrinkPoly 60 72 30 45

Computing at Entropy House

Just add Logo and stir...

Something of the functional style of TLC-Logo and LISP can be seen elsewhere in this issue. It's not hard to get used to it, but there is another form of programming becoming increasingly important, object oriented programming. We don't have the space to explain it here, but put simply, in an object oriented language you ask data to perform operations on itself, rather than commanding procedures to act on data. The original object oriented language is Smalltalk, and current debating item in LISP circles is a standard for object oriented LISPs. Coral Software's Object Logo for the Macintosh was due for release a couple of months back although it doesn't seem to have appeared yet. You can read about it, and object oriented programming in general, in last August's issue of *Byte*.

The ability of TLC-Logo to easily pass functions as arguments might sound esoteric, but it comes in useful at times. Imagine the Hilbert function on p 10 decently formatted and then compare it with this:

```
TO Hilbert :size :level :thisWay :thatWay
IF :level = 0 [STOP]
RUN LIST :thisWay 90
Hilbert :size :level - 1 :thatWay :thisWay
FORWARD :size
RUN LIST :thatWay 90
Hilbert :size :level - 1 :thisWay :thatWay
RUN LIST :thatWay 90
FORWARD :size
Hilbert :size :level - 1 :thisWay :thatWay
FORWARD :size
Hilbert :size :level - 1 :thatWay :thisWay
RUN LIST :thisWay 90
END
```

That RUN LIST... is the awkward part, @thisWay would be much easier. (Try Hilbert 10 3 "RT "LT)

There are times when COND would much neater than IF..THEN..ELSE, and to see what it does read the LISP version of Remove on p 10. You might try this sometime:

```
TO Cond :input
IF EMPTY? :input [PR "Oops! THROW "TOPLEVEL]
TEST RUN FIRST FIRST :input
IFTRUE [RUN LAST FIRST :input]
Cond BUTFIRST :input
END
```

This sort of thing works well enough...

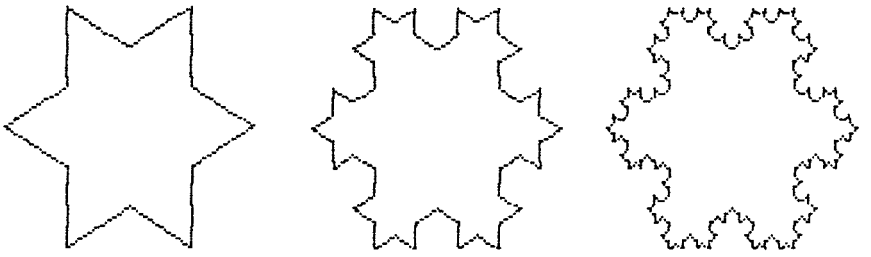
```
Cond [[[3 > 4][PR "four]][[6 < 5][PR "five]][[8 = 9][PR "nine]][["TRUE][PR "Truth!]]]
```

...but this doesn't:

```
PR Cond [[[3 > 4][OP "four]][[6 < 5][OP "five]][[8 = 9][OP "nine]][["TRUE][OP "Truth!]]]
```

One advantage of a functional Logo compared to a procedural one.

After the Hilbert procedure, Helge von Koch's curve. Put three together and you have a snowflake:



Level 1 Level 2 Level 3

The main procedure is the usual Logo recursive sort:

```
TO Koch :size :level
IF :level = 0 [FD :size STOP]
Koch :size / 3 :level - 1
LT 60
Koch :size / 3 :level - 1
RT 120
Koch :size / 3 :level - 1
LT 60
Koch :size / 3 :level - 1
END
```

```
TO SnowFlake :size :level
REPEAT 3 [Koch :size :level RT 120]
END
```

Compare that, and its results, with what's on p 14. David's Logo procedures are a direct translation of the BASIC. Moral: Your fractals will be as real as the language you use.

Another book that came our way recently deserves a mention, on this page note, and not Resources. It's *Computer Literacy An Integrated Course for Secondary Schools*, by R.E.D. and A.T.L. and published by Martin Educational. Chapter 3 is called 'Teaching LOGO' (*sic*) 'Originally', we are told, 'LOGO was designed as a graphical programming language. Since then other aspects ... have been added so that textual and arithmetical operations can be performed too.' (p 15) In fact, it was the other way about, and you can read that on p 218 of *Mindstorms*. The book goes on: 'The principles of teaching LOGO in this course are aligned with those of Seymour Papert, and expressed in his book, "Mind Storms, Children, Computers and Powerful Ideas."' They can't even get the name right. Have they read it one wonders?

Logo versions described are 'Apple LOGO, LOGO 2 (for the BBC computer) and OZ LOGO (for the Microbee)' 'LOGO 2' is from Computer Concepts in the UK, and noone around here knows anything about it. Anyone still using OZ Logo now that RL Logo is available is well behind it.

Chapter 5 is 'Assessment of Student Performance', complete with pretest, examinations etc., etc. On the disk of sample programs is one named MARKS, 'used to organise and adjust the marks from a test.' (p 26) Also on the disk is TRIANGL, a supplied triangle procedure. Where's the experimentation?

I'd often wondered about the difference between LOGO and Logo. Now I think I know. LOGO seems to be authoritarian computer literacy rubbish. Logo is different.

When the BBC's official AcornSoft Logo first appeared it was promoted as 'THE LOGO TO END ALL LOGO'S' (*sic*). (*Acorn Update*, December 1984) Acorn now bundles, as a ROM image on disk, *Logotron* Logo with new Compact machines. Most interesting. AcornSoft's Logo has its good points, but it's as slow as the proverbial wet week, has memory problems (like mode 2 being almost impossible), falls over with extended tail recursion and writes files that nothing else will look at. Logotron has its quirks too, but it's an LCSI type.

Registered users of LCSI/Microsoft Logo for the Macintosh would have received a note about the upgrade to version 1.1. It looks the same as 1.0, except for the background screen, but works properly with recent versions of the Finder, HFS and Mac Plus. It doesn't come with any documentation on how to set up the symbol space, stack size etc. for a megabyte of memory. Odd thing is, if you go to buy Logo you'll be handed 1.0 by default. If you want 1.1 you apparently have to ask for it. There are rumours of a version 2.

Seen an Apple IIGS yet? Think of the Logo that could run on it; all that memory, all the colours and 15 voice (ie. 32 channel) sound. SETPC 4023? Somehow I don't think it will work that way. Let's hope LCSI gets it right. As for the Macintosh II...

Research Logo Re-reviewed.

.. A reflection on Research Logo on the Premium Microbee.

David D. Curtis

I last put PD on the subject of Research Logo for *POALL* 1(4) in May 1986. At that time it was being used on the Microbee 128 which suffered the limitation of using programmable graphics characters. When all the available characters had been used, the turtle would continue to move but no further drawing would occur and the message 'OUT OF INK' would be displayed.

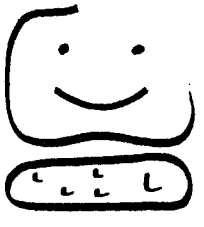
Microbee Systems have now released a new model, the Microbee 128 Premium. This machine still uses programmable graphics characters but additional screen RAM has been built in so that a full screen of graphics may be produced without the turtle running dry. To take advantage of the extended graphics features, a new version of Research Logo (Logo+) has been released. This is quite a big step FD. Other detailed improvments have been made to the machine so that the graphics are produced faster than before although the speed of the graphics display was never a problem.

For those people who have the earlier model Microbee, do not despair. Microbee Systems will upgrade your existing machine to Premium specifications. They have also made available an upgrade kit which is available through user groups so that the upgrade can be effected at a lower price for those with access to the necessary skills.

Microbee users should take an opportunity to .EXAMINE Research Logo as soon as they can. The availability of this improved version of Logo should make schools think more seriously about adopting this Australian designed and produced computer.

ACEC '87

We can look forward to these Logo papers at ACEC '87, plus some poster sessions and other events:



Burt, P. & Kazenwadel, W. *Adding a Logo dimension to Calculus*
Carter, P. J. *'...I've got a little list...'*
McDougall, A. *What do they learn when they're learning Logo?*
McMillan, B. *Logo and the teaching of Logo: Is it still Piagetian?*
Nevile, L. & Fox, C. *Are Microworlds Overrated?*
Nevile, L. *Logo is a language, so what?*
Tatnall, A. *Logo in control: Control technology in education*

What can I do?

Much of this issue, indeed, much of all recent issues, seems to be mainly for adults and older students. Here's something for younger readers. Some of these have been done, some of them haven't. Have a try at them. Send your results in and we'll publish some.