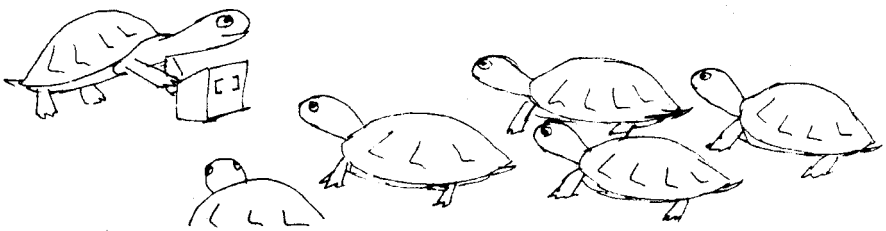
POALL

A Journal for Logo Users



Conference Reports

Volume 5 Number 1, November 1989

And so we begin Volume 5. How do you like our new look? It's an experiment, so do write and tell us what you think (and perhaps send an article)

Most of this issue is taken up with reports from the Australian Computers in Education Conference in Canberra and the Computers in Education Group of South Australia State Conference. Conferences are a time of sharing ideas, meeting new and old friends, and seeing new machines and software, and these two were no exception.

For once, there is no large Logo program.

Peter

Peter J. Carter, Editor

POTS

Address subscriptions and items for publication to: The Editor, *POALL*
Plympton High School
Errington Street
Plympton S. Aus. 5038

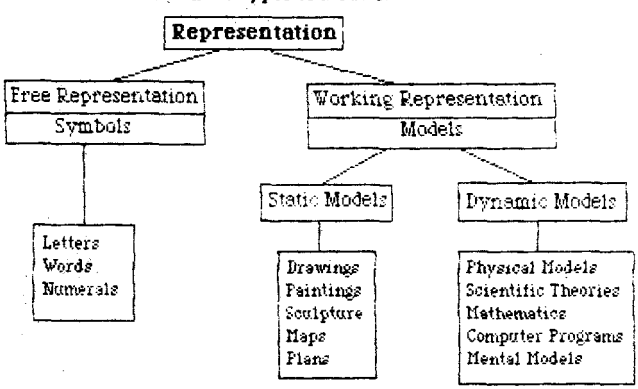
Australian Computers in Education Conference 1989

Logo was not a definite strand at this year's conference, and was virtually invisible in the exhibition, but was nevertheless in evidence. There was one clash in the program: Anne McDougall and Liddy Neville were both presenting at the same time (Boxer isn't Logo, is it.) LEGO/Logo was regularly demonstrated throughout the conference by Peter Smith. Published here are the abstracts of the Logo papers, and brief comments on the presentations.

Barry Newell: The Crowded Curriculum

Keynote speaker Barry Newell is well known for his Logo work and books *Turtle Confusion* and *Turtles Speak Mathematics* and his paper, 'The Crowded Curriculum', contains a section entitled 'The Importance of Programming' which describes some of the rationale behind the Turtle Trap problem outlined in *Turtles Speak Mathematics* and 'High School Mathematics: Logo and Linear Equations' (in the latest issue of *Australian Educational Computing*). Much of the paper concerns the representation of knowledge as models. To summarise, its headings:

1. Introduction
2. Some Basic Ideas
   (a) Prediction is a fundamental human activity
   (b) Prediction needs working models
   (c) We live in a world of 'representations'.
   (d) There are two main types of representations
      (i) B can represent A if we agree that it does
      (ii) B can represent A if B works in the same way as A in some essential aspect.
   (e) There are two main types of models.



3. Representation and Learning
   (a) Learning is a 'bootstrap' process
   (b) We learn by a 'Guess and Test' process
   (c) Learners must cope with a 'belief-disbelief dilemma'.
   (d) Learning is driven by a 'search for simplicity'.
   (e) Mental models are predominantly 'visual'.
   (f) Mental models are 'tacit'.
4. Two Teaching Problems
   (a) Our explanations are a limited source of 'model building materials'.
   (b) Our student's old konowledge is also a limited source of model building materials.

5. Representation of the curriculum
6. The Importance of Programming

'We will not make much progress, towards our goal of using computers to improve children's educational experiences, until we move beyond simple tool usage. If we wish to integrate computers into the curriculum, in more than a superficial manner, then we must begin to ustilise their fundamental properties. The representational approach advocated here suggests that, in essence, computers are devices for building working models of the world. These models are called 'programs'. Thus, building and running computer programs is one of the most direct ways that children can gain first-hand experience in constructing and using articulated representations. (p 43)

7. Conclusion

Margaret Kennedy: Beginner's guide to Logo

'Introducing LOGO (*sic*) to less able children, Margaret discovered the need for a workbook to help these children acquire the basic framework to enable them to print, save, etc. She therefore wrote her own workbook, constantly revising it to suit the particular needs of the children. The workbook will be available for people to try out.'

Margaret outlined a number of problems in using Logo in the classroom...
1 Logo is not as easily integrated as adventure games, and students cannot work as independently.
2 Children need help at hand, eg. to understand error messages.
3 Teachers, often new to computing and Logo, need support.
4 Children need to work at their own pace, and keep records of successes and failures.
5 Scheduling machines and teachers can be difficult.
... and reasons why Logo is a valuable learning experience.
1 Logo is fun.
2 Drawing is a logical extension of an everyday activity.
3 Logo allows all children to use their creativity and achieve a satisfactory level.
4 Logo is an excellent tool for the intellectually gifted.
5 Logo encourages problem solving and discovery.

She demonstrated aspects of the workbook (based on Logotron Logo), designed to provide the basic commands, a range of activities and reinforcement and a springboard to further activities. Among activities shown were screen mazes. All of us working with Logo have been down similar paths, but many in the audience were clearly new to Logo, and for them it was a useful presentation.

Pam Gibbons: Confusion, Gognition and Metacognition (The agony and the ecstacy)

'During 1988 I made use of Barry Newell's book *Turtle Confusion* as a resource in teaching a tertiary course on Logo and problem solving. It is perhaps fitting that I would like to use this conference, at which Barry is a keynote speaker, as a forum to discuss the results and implications of the venture. The project has raised some important issues in my own mind about the hurdles that will need to be overcome if problem solving is to be a serious curriculum objective in Australian schools. This is not a session for those who are looking for hard-core research; I hope it will be more of a 'fireside chat' about what happened when I put an unusual resource into action with a group of hitherto traditionally educated adolescents.'

Barry Newell was in the audience, and we were given the task of solving two of the puzzles (1 and 31) from *Turtle Confusion*.

Pam then showed a number of solutions devised by members of her class at the Catholic College of Education in Sydney. The paper itself contains a number of comments from the students, one observing that he had never before been asked to solve a problem that he did not already know how to solve.

Pam concludes: 'If we wish problem solving to have an impact in our schools, we must provide students with opportunities to gain experience in the solving of true problems. If metacognition is a valuable tool in true problem solving, then perhaps we should not only provide opportunities for students to gain experience in describing their own metacognition, but we should assist in the development of appropriate language to do so.

An attitude which does not narrow correctness down to one solution, the skill of productive lateral thinking, the ability to analyse ones own thinking and the confidence to try new approaches are facets of true problem solving which would have a greater chance of bearing fruit if they were nutured in children from an early age.'

Peter Carter: Stepping out: Legged robots in LEGO

'We take walking for granted, but legged robots are the subject of intense study at various research intsitutions around the world. With LEGO/Logo it is possible to build simple legged machines and explore some of the issues and problems.'

You read most of this in a previous issue of *POALL*, but illustrated this time with slides, and with the inclusion of two new leg mechanisms (see pp 6 & 19). The machines and second projector all worked, and the presentation was later described by one person as 'laid back'.

Anne McDougall: Recursions in Logo

This paper reviews research on teaching recursion in Logo programming, and resources for this. It considers some areas of difficulty described in the research literature, including treatment of only tail recursion, encouragement of looping models for recursion, effects of previous experience in BASIC programming, and that teachers and authors may be presenting a topic that they do not fully understand. It notes that these problems are reflected in inaccuracies and confusing treatments in some books on Logo.

Anne presented examples of correct models and explanations of recursive processes, many of them from her PhD research work. Question time at the end brought forth a question about the need for recursion, since recursion was rarely used in Pascal, with its choice of looping constructs.

Some of these issues are explored in an item elsewhere in this edition of *POALL*.

Jeff Richardson: Computer Environments as media for expression (a polemic)

'A computer environment is a medium. So is televison. So is painting. So is print or writing. Or are they? It is a commonplace remark that the *information explosion* or the *silicon chip revolution* is as significant as the spread of the printing press five centuries ago. Now writing existed before the printing press, but Ivan Illich has argued (*ABC, the alphabetisation of the human mind* North Point Press 1988) that printing and the spread of what we might crudely call literacy has changed the way that people think.

Strangely perhaps, a similar claim does not seem sustainable for moving pictures, even though their relationship to still pictures is much more dramatic. (Marvin Minsky, quoted in Stewart Brand's *The Media Lab* (Penguin 1988) said "Imagine if TV was actually good! We'd have to rethink everything ")

A similar claim is made however, for computer environments, by Seymour Papert (*Mindstorms* 1980).

I will argue that computer environments offer the possibility of engagement to writers **and** readers (in a broad sense) in a way that writing does and TV does not.

Great artists continue to create enduring works of literature, while the ordinary person continues to use writing and printing to create and publish verbal objects. Since the inventions of both cinema and TV, some canonical works have been created. The ordinary person however is almost totally cast as a reader, or consumer, even though video technology appears to offer something different. Similarly, the software industry, while less than fifty years old, has thrown up very few masterpieces. Within the computer culture however, there have been a lot of ordinary people doing important creative work.

Computer use in education can be divided into approaches which place the learner as a reader, a writer or both. It is possible to read and analyse computer environments as texts. It is only possible however to use **some** computer environments for writing, creating, further computer environments, or texts.

I wish to emphasise the use of computer environments as media for creative expression. Such environments may be the more commonplace word processors, programmable spreadsheets, databases or integrated works style environments. Or they may be richer, more open ended and slightly exotic environments like LogoWriter, Boxer or HyperCard.

The argument will be presented very briefly and the session given over to discussion from the floor.'

Discussion from the floor was quite lively, particularly at several of Jeff's comments: 'Most of the software here [in the exhibition] I despise.' and 'Carmen Sandiego? I'd rather read the atlas.' Another was that there were only two creative pieces of software, LogoWriter and HyperCard. Obviously not everyone's view, but then, what is creativity? How can one be creative with *Green Globs*?

Liddy Neville: Introduction to Boxer

Since the published abstract was for a different paper, a couple of extracts from the paper itself:

'Boxer is not just a smarter or faster version of existing systems. Boxer has its own unique design features which have been carefully developed to make the human/computer interface as intuitive as possible. The spatial metaphor which is the basis of the environment is native to Boxer (although it is now appearing in other systems). The hyperspace qualities of Boxer extend beyond the capabilities of standard hypertext systems, and the boxes of Boxer are not cosmetic but fundamental to the system. ...

Boxer does not fit into this category [of normal information handling packages]. Boxer is a working environment for thinkers. It has evolved in the rarified atmosphere of the academic world. Unless there is a concerted effort to bring this technology out into main stream education, this type of electronic support of thinking for learners will not necessarily exist. If Boxer does get this support, it will be suitable for a wide range of learning situations including learning about the established world and inventing the future world.'

Computers in Education Group of South Australia State Conference, October 20..22

The conference itself was held at SACAE Magill, but the opening on the Friday evening was at the Technology School of the Future at The Levels. Garth Boomer was opening speaker, and presented ten axioms and warnings about technology in education. A selection: 'The human brain learns experimentally, and is aggressive to learn.' 'Brain power is shut down when someone else takes responsibility for learning.' 'Students should demand and expect to use computers as tools.' 'Most testing is a brain hazard.' 'Schools are dominated by words' (and not graphics, music etc. ie. multimedia) The whole emphasis was on designing and planning.

Rosemary Williams, of Portland University, Oregon, was keynote speaker on the Saturday, with the topic 'Changing the way we think: Technology in education and everyday life.' The theme was a quotation from James Dickey:

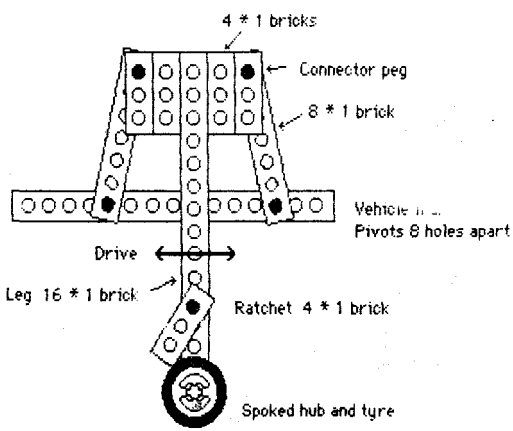
> 'One constantly hears the computer referred to as a tool, as though this were reassurance of some sort. It is reassurance only until one remembers how the tool has shaped the human hand, and notes with a shock that this tool is shaping not the hand but the mind. A tool used as extensively as the computer cannot help influencing how we think.'

Again, some good constructivist stuff, with emphasis on learning in a culture, by tinkering and experiment.

There was only one Logo paper, published on the next pages, but Irene Meyer ran two well attended LEGO/Logo workshops.

More LEGO Legs

LEGO walking machines, because of the limitations of sensing and control, must be statically stable. For a quadruped, that means either that only one leg can be in the air at a time, or the machine keeps all four on the surface at all times and shuffles. The leg shown below maintains a constant length as it moves through a small arc. Almost any gait can be used with it.



(continued on p 19)

recursion *n.* (*see* recursion)

Abstract: Without a sound understanding of recursion it is difficult, if not impossible, to progress with Logo. Perhaps some of the difficulties people face are caused by misconceptions about what recursion is and is not. This paper looks at some of those misconceptions, and suggests more satisfactory explanations of the process.

'Recursion is the act of defining an object or solving a problem in terms of itself. A careless recursion can lead to infinite regress. We avoid the bottomless circularity inherent in this tactic by demanding that the recursion be stated in terms of some "simpler" object, and by providing the definition or solution of some trivial base case. Properly used, recursion is a powerful problem solving technique, both in artificial domains like mathematics and computer programming, and in real life.'

The words of Friedman and Felleisen in their preface to *The Little LISPer* (1987), an amusing LISP text with a clear emphasis on recursion. It is not necessarily the purpose of this paper to try to convince you that recursion is a 'powerful problem solving technique', but to give some familiar examples of recursion, and to give an explanation, by analogy, that may help to explain what happens during the execution of recursive Logo procedures.

Whether we realise it or not, recursion is common in our lives. News broadcasts and telecasts frequently include segments from reporters outside the studio, and in turn, outside reporters often present other speakers. We have no difficulty keeping track of the report (within the report) within the report. Modern telephone systems allow us to put a caller on hold while we make another call to find some information before returning to the original conversation. Language itself contains recursive elements, with clauses and phrases nested within sentences, and this is more apparent in some languages, German for instance, with verbs at the end of sentences, than in others.

In the arts, music, with often subtle key changes, shows recursive structure, although unless the listener either has perfect pitch or can follow the score, the level of nesting is easily lost. Shakespeare's 'Hamlet', with its play within the play, is but one example of recursive theatre. Most people are familiar with Russian dolls, nested one inside the other, and recursive pictures are quite common. It was just such a picture on a book cover that disturbed Sherry Turkle (1984):

'Whenever I looked at the photograph or the book I couldn't stop thinking about them, yet could find no way to capture for myself or for anyone else exactly what it was that was so upsetting and gripping for me.

Other children meet this experience in the form of questions about where the stars end or whether there is ever a final image when mirrors reflect mirrors. In all of these cases, what disturbs is closely tied to what fascinates and what fascinates is deeply rooted in what disturbs.

When I was in trouble with self-referential pictures I could get no help. The adults around me were no better able to handle the infinite series of ever smaller little girls than I was, except to assert their authority by telling me not to think about such things. Children's encounters with ideas like self-reference, infinity, and paradox are disturbing and exciting and are made all the more mysterious by the fact that appeals to parents about them are likely to provoke frustrating admonitions not to think about such slippery questions. Yet such questions become storm centres in the mind.' (p 23)

Other workers have found, as did Turkle, that quite young children are able to recognise recursion and self-reference. Some children's literature, for instance *The Cat in the Hat comes back*, contain recursive elements. But there are more subtle recursions in nature. Benoit Mandelbrot has drawn attention to the 'fractal' nature of much of the world, and fractals since have become a popular means of

demonstrating recursion. William Poundstone's thesis (1985) is that the universe can be described in a few simple, recursive rules, demonstrating his case with the game of 'Life' and other cellular automata. (Brian Silverman has, in effect, turned the book into software with *The Phantom Fishtank*.) Recursion in some form is inescapable.

Logo and recursion are almost synonymous, and there are occasional arguments about whether a programming language for students should use recursion so widely or be equipped with looping structures: REPEAT...UNTIL, FOR...DO, WHILE...DO, etc. (Logo's REPEAT is less powerful than those.)

More than once I have listened to discussions in the Pascal community about the value of both REPEAT...UNTIL and WHILE...DO, because students tend to be uncertain about which to use in which circumstances. Abelson, Sussman and Sussman (1985) are characteristically blunt in their comments about such choices, and their machine implementation: '...special iteration constructs are useful only as syntactic sugar.' (p 33) In other words, there need be no choice, iteration can be performed more effectively with tail recursion.

Many writers of Logo books, especially in the early years of Logo, offered explanations that were plainly wrong. That may have been due to their unfamiliarity with Logo compared with, say, BASIC, but the impression they leave is false. One example, by Gruber (1983):

> 'This brings us to what I would consider a more controversial aspect of the language: recursion, that peculiar programming construct in which a procedure calls itself. Here's a very simple example... [With a line missing in the original]
>
> ```
> TO PRINT.TO.10 :NUM
> IF :NUM > 10 [STOP]
> PRINT.TO.10 :NUM + 1
> END
> ```
>
> ... The lines are executed in order, just as in BASIC. ...
> The key is the next line...
>
> ```
> PRINT.TO.10 :NUM + 1
> ```
>
> This is the recursion, the procedure calling itself but with the argument :NUM increased by 1. Actually, in this simple example, the recursion is exactly equivalent to a GOTO the IF line.' (p 14)

The fact is that the recursive call is *not* equivalent to a GOTO, and the procedure does not call 'itself'. Elsewhere in the article, Gruber discusses local variables, but nowhere does he offer a clear explanation of recursion. One is left with the impression of a loop. Recursion is *not* looping. Recursion is nesting, interruption and deferral; procedures calling new copies of themselves.

Other writers of the same period offer descriptions similar to Gruber's, or no explanation at all. Heller *et al*, for example, have six pages devoted to very cumbersome fractal procedures, but make no attempt to describe or explain the process, and the word 'recursion' does not appear in the index.

Fortunately, other writers, who seem to have come from a mathematics/computer science/LISP background have been more helpful. The most accessible writer is Brian Harvey, whose *Computer Science Logo Style* trilogy should be required reading for teachers, and course planners and administrators. In Volume 1 (1985) he presents four Chapters 5, each dealing with recursion from a different viewpoint: the combining method, the little people method, the tracing method, and the leap of faith method. I would suggest that you read these at your leisure.

Let me offer an explanation that is similar to Harvey's 'tracing method', but embellished by ideas adapted from Douglas Hofstadter, and taken from *Thinking Logo* (Carter 1987, pp 16..17). For a procedure, a small numerical example:

```
TO Counting2 :number
IF :number = 0 [STOP]
Counting2 :number - 1
PRINT :number
END
```

'Turtles are known to occasionally mutter to themselves as they work. One was once overheard as it worked on this problem: (Seems it had access to a photocopier...)

"Hmm, Counting2 3. :number's not 0, so what's next? Counting2 3 - 1." (Makes copy of Counting2, writes 2 on it, and puts Counting2 3 on the table.)

"Hmm, Counting2 2. :number's not 0, so what's next? Counting2 2 - 1." (Makes copy of Counting2, writes 1 on it, and puts Counting2 2 on Counting2 3 on the table.)

"Hmm, Counting2 1. :number's not 0, so what's next? Counting2 1 - 1." (Makes copy of Counting2, writes 0 on it, and puts Counting2 1 on the stack.)

"Hmm, Counting2 0. :number is 0, so that's the end of that one!" (Tosses it into the bin and picks up the top copy from the pile on the table.)

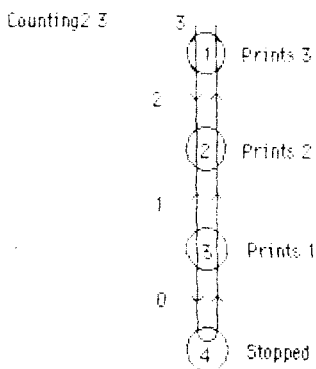
"Ah yes, print the value of :number." (He writes a 1 on the screen, drops Counting2 1 into the bin and picks up the top copy.)

"Ah yes, print the value of :number." (He writes a 2 on the screen, drops Counting2 2 into the bin and picks up the top copy.)

"Ah yes, print the value of :number." (He writes a 3 on the screen and drops Counting2 3 into the bin.)

"Done." he murmurs with a satisfied grin.

In many instances of recursion a process is replaced by a modified copy of itself; PolySpi is a classic example. In others, like Counting2, the process is deferred until the simplest case is finished and the recursion 'unwinds'. Below is the first of many diagrams showing how recursive processes work. The numbered circles represent calls to the procedure, 4 in this instance. The downward pointing arrows on the left show values being passed. If there were outputs they would be shown upwards on the right but Counting2 simply prints its values.



The choice of the word 'stack' is deliberate. The Logo interpreter uses a stack to store the variables for each instantiation of the procedure: at each call, the values are 'pushed' to the stack, as the recursion unwinds, they are 'popped' off. When a Logo system crashes with an OUT OF MEMORY error the problem is really a stack overflow, usually caused by a recursive procedure without an adequate stop rule. Mention of the bin in the story is an allusion to 'garbage collection', the method by which the interpreter reclaims unused memory.

Most modern Logo implementations have a TRACE facility, which allows inputs and outputs, and the level of recursion, to be monitored. A list processing example:

```
TO Remove :item :list
IF EMPTY? :list [OUTPUT []]
IF :item = FIRST :list [OUTPUT Remove :item BUTFIRST :list]
OUTPUT SENTENCE FIRST :list Remove :item BUTFIRST :list
END

SHOW Remove "potato [apple orange banana potato pineapple apricot]
 Remove potato [apple orange banana potato pineapple apricot]
  Remove potato [orange banana potato pineapple apricot]
   Remove potato [banana potato pineapple apricot]
    Remove potato [potato pineapple apricot]
     Remove potato [pineapple apricot]
      Remove potato [apricot]
       Remove potato []
       Remove Outputs []
      Remove Outputs [apricot]
     Remove Outputs [pineapple apricot]
    Remove Outputs [pineapple apricot]
   Remove Outputs [banana pineapple apricot]
  Remove Outputs [orange banana pineapple apricot]
 Remove Outputs [apple orange banana pineapple apricot]
[apple orange banana pineapple apricot]
```

It is not difficult to draw a diagram from such a trace, but the task becomes more interesting when the recursion branches..

```
TO CountWords :object
IF EMPTY? :object [OUTPUT 0]
IF WORD? FIRST :object [OUTPUT 1]
OUTPUT (CountWords FIRST :object) + (CountWords BUTFIRST :object)
END

PR CountWords [This is [a list][of lists]]
```

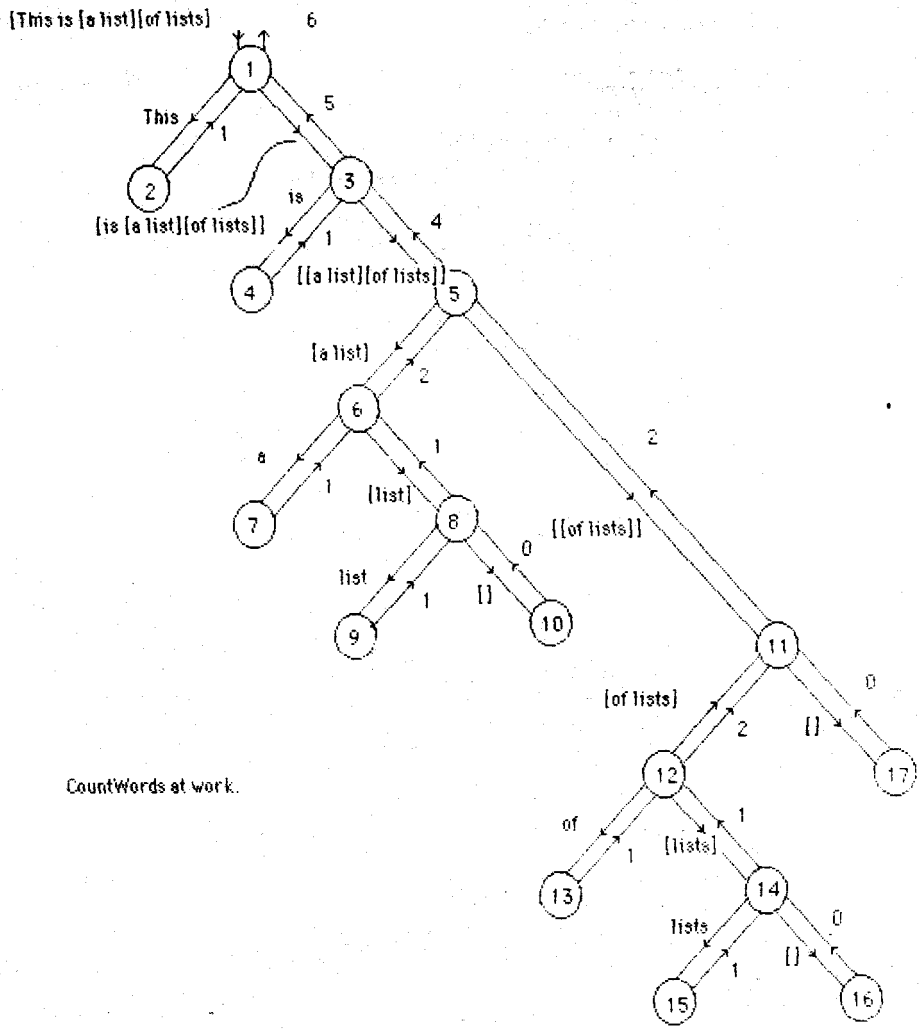
(see diagram on next page)

How should recursion be introduced? Most writers begin with tail recursion (PolySpi etc.) But there is a body of opinion that suggests that 'embedded' recursion be used first, with tail recursion coming later as a special case, this sequence reducing the likelihood of recursion being seen as looping. I generally begin with tail recursion, but always with stop rules, and emphasise that the procedure is not calling itself, but a new copy of itself. A procedure like this can be used...

```
TO Steps :length
IF :length < 10 [STOP]
FORWARD 10 RIGHT 90 FORWARD 10 LEFT 90
Steps :length - 10
END
```

...and then changed to.

```
TO Steps :length
IF :length < 10 [STOP]
FORWARD 10 RIGHT 90 FORWARD 10 LEFT 90
Steps :length - 10
RIGHT 90 FORWARD 10 LEFT 90 BACK 10
END
```



CountWords at work.

How well do students cope? Turkle (1984) cites the case of 'Matthew' who was able to understand recursive procedures at age five. I have seen eleven and twelve year olds cope with tail recursion, but on the other hand I have seen Year 11 and 12 students in difficulties (and not only with recursion. I question the value of current Computing Studies courses for many students). John de Figueirdo (1989) found many of his students in difficulties. I have a strong suspicion that many of the problems can be overcome with adequate descriptions and explanations of recursive processes.

How does one design recursive procedures? Perhaps the advice of Hofstadter (1986) is the most concise:

> '...To spell out the exact nature of this recursion-guiding pathway, you have to answer two Big Questions:
>
> (1) What is the embryonic case?
> (2) What is the relationship of a typical case to the next simpler case?
>
> Now actually, both of these Big Questions break up into two subquestions (as befits any self-respecting recursive question!), one concerning how you recognise where you are or how you are to move, the other concerning what the answer is at any given stage. This, spelled out more explicitly, our Big Questions are:
>
> (1*a*) How can you know when you've reached the embryonic case?
> (1*b*) What is the embryonic answer?
>
> (2*a*) From a typical case, how do you take exactly one step toward the embryonic case?
> (2*b*) How do you build this case's answer out of the "magically given" answer to the simpler case?
>
> Question (2*a*) concerns the nature of the *descent* towards the embryonic case, or bottom line. Question (2*b*) concerns the inverse aspect, namely, the *ascent* that carries you back up from the bottom to the top level.' (p 416)

Long ago (in computing terms) LISP was given the power of recursion for the symbol manipulation of AI research. Logo has that power without the confusing syntax and unnecessary choices of looping constructs of other languages. Logo is a language which allows students the freedom to explore and be creative, with recursion as a perfectly natural part of the language and problem solving techniques.

One is mindful, however, of Hofstadter's Law: 'It always takes longer than you expect, even when you take into account Hofstadter's Law.'

References

Abelson, H. and Sussman, G. with Sussman, J. *Structure and interpretation of Computer Programs* MIT Press, 1985

Carter, P. J. *Thinking Logo* 1987

de Figueirdo, J. 'Students don't think recursively' *POALL* Vol 4 No 4, pp7..15

Friedman, G. and Felleisen, M. *The Little LISPer* MIT Press1987

Gruber, A. 'From LISP to Logo' in *Call A.P.P.L.E.* Vol 6 No 8 August 1983

Harvey, B. *Computer Science Logo Style: Intermediate Programming* MIT Press 1986

Heller, R., Martin, C. and Wright, J. *LOGOWORLDS* Computer Science Press 1985

Hofstadter, D. *Godel, Escher, Bach: An Eternal Golden Braid* Penguin 1980

Hofstadter, D. *Metamagical Themas: Questing for the Essence of Mind and Pattern* Penguin 1986

McDougall, A. 'Teaching about Recursion in Logo: a Review' in Dupé, T. *ACEC '89 Backup the Future* CEGACT 1989

Poundstone, W. *The Recursive Universe* Wm Morrow & Co 1985

Silverman, B. *The Phantom Fishtank* LCSI 1987

Turkle S. *The Second Self: Computers and the Human Spirit* Granada 1984

Diagramming Recursion

As part of the presentation of the above paper, the Counting2 procedure was diagrammed on screen, by (naturally) a recursive procedure:

```
TO BestC2 :number
CLEARTEXT
BestC2Aux :number :number 0
END

TO BestC2Aux :level :number :tab
Tab :tab TopLine 16
Tab :tab PR (SE [|TO Counting2] :number [\ |])
Tab :tab PR (SE [|IF] :number [= 0 \[STOP\]  |])
IF :number = 0 [Tab :tab BottomLine 16 STOP]
Tab :tab PR (SE [|Counting2] :number - 1 [ \ \ \ |])
BestC2Aux :level :number - 1 5 * (1 + Abs (:number - :level))
Tab :tab PR (SE [|PRINT] :number [ \ \ \ \ \ \ \ |])
Tab :tab PR [|END\ \ \ \ \ \ \ \ \ \ \ \ \ |]
Tab :tab BottomLine 16
END
```

```
TO Tab :spaces
REPEAT :spaces [TYPE CHAR 32]
END

TO BottomLine :length
TYPE "|
REPEAT :length [TYPE "_]
PRINT "|
END

TO Abs :number
OUTPUT IF :number < 0 [- :number][:number]
END

TO TopLine :length
TYPE CHAR 32
REPEAT :length [TYPE "_]
PRINT CHAR 32
END
```

```
 _____________
|TO Counting2 3  |
|IF 3 = 0 [STOP] |
|Counting2 2     |
      _____________
     |TO Counting2 2  |
     |IF 2 = 0 [STOP] |
     |Counting2 1     |
          _____________
         |TO Counting2 1  |
         |IF 1 = 0 [STOP] |
         |Counting2 0     |
              _____________
             |TO Counting2 0  |
             |IF 0 = 0 [STOP] |
             |________________|
         |PRINT 1         |
         |END             |
         |________________|
     |PRINT 2         |
     |END             |
     |________________|
|PRINT 3         |
|END             |
|________________|
```

You can adapt the idea to any recursive procedure.

Resources

Commodore Amiga Logo

A Logo has just been released for the Commodore Amiga. It's written by Carl Sassenath for Commodore-Amiga and is designed to emulate Apple Logo, even to the Control key commands. That's a curious decision; here is a 1989 version of Logo, running on a 68000 machine, and emulating a limited 1982 6502 implementation. That imposes a number of restrictions.

Turtle is turtle shaped, small, and green and gold in colour. There can be 31 colours on the screen at once, and the palette can be changed with the SetRGB primitive; pencolours can be changed after objects have been drawn. In the default palette, colours 1..6 are those of Apple Logo. Colour 0 is always the background colour. There's a Fill primitive, and also FillIn, which fills areas drawn in any pen colours, and a GRType, which doesn't erase like that in LCSI Logo II. Dot is implemented, but not DotP.

An interesting feature is Mouse Draw. Turtle is dragged about by the mouse, so freehand drawing and Turtle graphics can be mixed.

The graphics area always fills the whole screen, with the text or editor window superimposed on it. The text window can be moved or resized. TS, SS and FS work as expected. Typing TO ... takes one to the edit window, which one leaves with ‹CTRL› C or ‹CTRL› G. There is an EditFile, but to save the file one leaves the editor with ‹CTRL› G and uses SaveFile. Self starting files must be generated with EditFile by adding a command line at the end, rather than with :STARTUP (A similar technique can be used with View and Logotron on the BBC.)

Property lists and buried packages are implemented. Predicates may end in either P or ?, eg EmptyP or Empty?.

The manual is in a ring binder, in two sections, tutorial and reference. There is an index, but no bibliography, and no real technical detail on the implementation. Although the manual suggests that the version was designed to make it easy to use with material from existing Logo books, none is listed. To go with the package, Commodore Australia has commissioned some beginners' material from Pam Gibbons in Sydney. She was happy to write it, but puzzled.

The other main area of interest is the Say primitive. Assuming the machine has Workbench 1.3 and the necessary hardware, Logo can generate speech.

So much for what Amiga Logo has. Now for what it hasn't. There are no multiple or redefinable Turtles or sprites. In that regard, this Logo is less capable than Commodore 64 Logo of 1983. Neither Step nor Trace is implemented, so debugging support is limited. (Will there be an Amiga Logo Tool Kit like the old Apple Logo Tool Kit to supply the necessary procedures?)

The Logo seems to run at a respectable speed, about 8 seconds for a 200 line PolySpi, but each run of the procedure needed at least one, if not two, garbage collections. That's interesting, because tail recursion shouldn't slow things down. I was unable to crash it with 50 level list processing recursion, so at least there should be no problems in that regard.

There is one feature (?) that this reviewer finds particularly annoying. Type PR "fred and this Logo returns FRED, in other words, although it is case insensitive as regards input, Amiga Logo output is always all capitals. (In my view, text in all capitals deserves capital punishment.) Something else is definitely a bug. With this procedure (How I write it, not how Amiga Logo prints it):

```
TO Remove :item :list
IF EMPTY? :list [OP []]
IF :item = FIRST :list [OP Remove :item BF :list]
OP SE FIRST :list Remove :item BF :list
END
```

... this input:

```
show remove "z [a b c d e f g h i j k l m n o p q r s t u v w x y z]
```

returns:

```
[A B C D E F G H I J K L M N O P Q R S T U V W X Y [ ]]
```

In other words, Sentence is tacking empty lists and words on to objects. That's not what Logo should do, and it could lead to all sorts of interesting problems.

Another potential problem is that one cannot have a procedure and a free variable with the same name. That's easy enough to program around, but it's an unnecessary restriction.

It's good that the Amiga now has a Logo, but it's a strange one, and one could be forgiven for wondering why Commodore didn't have a version written by Terrapin or LCSI, instead of by someone who is clearly outside the mainstream of Logo. I can't help thinking it might be better to use the Amiga's MS-DOS capability to run LogoWriter.

PC Logo

PC Logo, by Harvard Associates, has been around for a while, and version 3.0 is now available. PC Logo makes much better use of the PC's features than does Amiga Logo of the Amiga, to the extent of using an 8087 math coprocessor if fitted, access to the BIOS, and the capacity to use either CGA or Hercules standard graphics. Syntax is MIT/Terrapin style, with predicates ending in ?, and IF..THEN..ELSE rather than IF..[..][..]. Stepping, tracing and other debugging primitives are included, as is the ability to work in bases other than 10. Playing with binary or hexadecimal numbers is easy, and there are logical operators, LOGAND, LOGOR, etc. for use with binary numbers. Some aspects of computing science are therefore easily demonstrated. There is full access to files.

On a NEC Powermate 1 Plus, a 200 line PolySpi took about 3.5 seconds, even with the triangular Turtle visible, and quite smoothly, without any garbage collections. List processing was also quite fast. It's possible to set stack and other memory sizes at startup, so there should be few problems with stack overflows. However, like Amiga Logo, this one also returns objects in upper case. Normally, PC Logo is case insensitive, but the CASE and NOCASE primitives change it to case sensitive. There seems to be little excuse for this sort of behaviour. If a system needs, to save memory, to store procedure definitions in all capitals, that is something that can be lived with, but to arbitrarily change text and other objects seems quite unnecessary.

PC Logo comes with a tutorial manual, a reference manual and a quick reference guide; comprehensive and well written. There is also a utilities disk with some sample programs (the old DYNATRACK, ANIMAL etc.), and an order card for the MIT Logo books (Harvey *et al*) and books by Birch. The package is $199, and site licences are available. For Year 12 students it would be a good system, for younger students I'd still prefer LogoWriter. My thanks to Terry Malone of EdSoft for the opportunity to try it.

Turtles Speak Mathematics

Turtles Speak Mathematics is Barry Newell's second Logo book. Like *Turtle Confusion*, the book is in the form of a dialogue between EBN and the Turtle, this time without puzzles and riddles, but with discussion of many issues relating to the teaching of mathematics and general problem solving:

> "'...my question is "how does a real-life, hardworking, syllabus-bound teacher make use of mathematics-speaking turtles?".'

'That's the central question,' said the Turtle. 'That's **the** central question..."

Logo's place in the curriculum has always been problematical, and Newell's answer is clear: Logo is a notation for problem solving in many aspects of mathematics.

"'What do you real by "real mathematics"?' said the Turtle, with a twinkle in his eye. 'The mathematics of real numbers?'

'I mean things like algebra and trigonometry and coordinate geometry and calculus and estimation and statistics and ...'

'Good!' said the Turtle, rubbing his hands together and holding them out to the fire. 'You can meet all of those topics with the turtle's help ... but you need to add iteration and recursion and topology and differential equations and physics and animal behaviour and artificial intelligence and robotics and principles of design ... the list is almost endless.'"

The main part of the discussion is centred around the Turtle Trap, procedures based on some ideas from Chapter 2 of Abelson and diSessa's *Turtle Geometry*. The idea is to restrict the Turtle to one part of the screen as it wanders RANDOMly about.

Equally central to the book is a parable against 'disembodied learning', caused by artificial boundaries between subjects.

The book concludes with a list:

Logo provides:

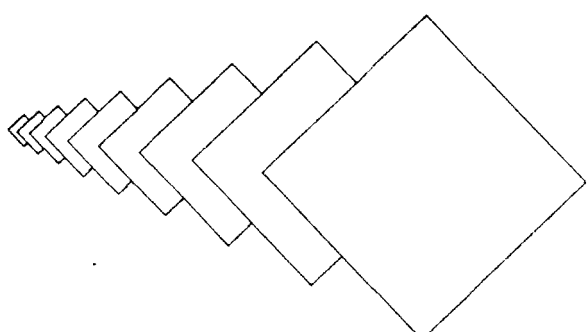
1. contact with fundamental and useful mathematical ideas.
2. interesting contexts that give reasons to learn the mathematical ideas.
3. links (between intuitive knowledge and formal ideas) that are the means to learn the mathematical ideas.
4. working models of mathematical ideas and scientific concepts.
5. good feedback so that you can assess your own ideas and understanding.
6. first-hand experience with the power of formal mathematical knowledge.

Turtles Speak Mathematics should be read by all teachers of mathematics.

Others

Two potentially interesting books have just been reviewed in *Nature*. The first is *The Turing Omnibus: 61 Excursions in Computer Science* by A. K. Dewdney, who writes the 'Computer Recreations' column in *Scientific American*. Publisher is Computer Science Press (distributed by W. H. Freeman), 415 pages for $US24.95.

The other is *Turtles of the World* by C. H. Ernst and R. W. Barbour. It's a review of the 257 species and 200 million year history of the turtle, and is a publication of the Smithsonian Institution. Price is $US45.



Pascal, Logo, or something else?

Why teach Pascal in schools? That was the main question asked in 'The Pascal Experiment' as it was published in *POALL* or 'Pascal? Why?' in an abbreviated and serialised form in the CEGSA Newsletter. So far, noone has answered.

Yes, one was 'stirring', but there is a serious intent, and there are some other questions that perhaps need thinking through. Why teach programming? Why the emphasis on data processing? Why not skills of planning, modularity etc. but without the rigid analysis/design/code/validate scheme? Why not programming by experiment and debugging? Why not OOPS and hypertext, the current 'in' paradigms? What do the universities and SAIT want of their prospective students apart from keyboard skills and the ability to use word processors etc? Do they really want familiarity with Pascal syntax, or notions of planning and modularity? What about our students? Are we aiming at future computing professionals, at computer users in other professions and trades, or at students who have run out of Tech. Studies and Home Ec. options and don't have the intellectual capacity to cope with the abstract and symbolic notions of programming? (NB. I'm not disparaging Tech. Studies and Home Ec.) Of course there may be aspects of High school that this former Primary school teacher has yet to understand.

That a student's first programming language is influential has been observed by many writers. Two examples:

> 'Our experience, and that of others who teach programming, is that a first computer language's particular style and its main concepts not only have a strong influence on what a new programmer can accomplish but also leave an impression about programming and computers that can last for years. The process of learning to program a computer can impose such a particular point of view that alternative ways of perceiving and solving problems can become extremely frustrating for new programmers.' (Kay, 1977)

> 'Along with this dissatisfaction [about existing languages] goes my conviction that the language in which the student is taught to express his ideas profoundly influences his habits of thought and invention, and the disorder governing these languages directly imposes itself onto the programming style of the students.' (Jensen and Wirth, 1975)

Given that, what are the qualities needed in a language for students, and by students I mean those at primary and secondary level? Perhaps the following, in no particular order:

- Interactive, either interpreted or incrementally compiled.
- Easily used editing and filing systems.
- Easily used graphics (Turtle and coordinate) and sound. ('Sprites' and 'demons' would be useful, but not essential.)
- The ability to control external devices (eg. Turtles and LEGO machines).
- Modular, with two-way parameter passing.
- Minimum syntax, and consistency in forming constructs.
- Convenient, but not restrictive data typing.
- Meaningful error messages.
- Easy manipulation of text.
- Easily used looping constructs, and recursion.
- Mathematical functions.
- Power and expressiveness.
- The ability to demonstrate and explore fundamental issues in computing.

How many languages meet those criteria? There was some interesting correspondence, following an article on the subject, in Byte in 1984. One Wendell Brown wrote almost despairingly (Brown, 1984):

> 'Are we to go on teaching bad habits to our beginning programmers? Pascal is certainly an alternative. But Pascal is intimidating and hard for many students to learn as their first language. Editing, compiling, executing, and re-editing to debug a program can severely test the patience of a young person...
> Isn't it about time somebody offered a well-structured, incrementally compiled language for the Apple II?'

There were several replies, one from an obvious Logophile, who after describing Logo's features stated (Teller, 1984): 'The language is Logo.'

Now, Logo is not without its faults. Andy di Sessa has admitted that from the time the idea of the Turtle appeared, list processing received less attention than it should have in the design of Logo. We still have much of the power of LISP, but it is often, to some people, inaccessible. To write a large and complex list processing program requires considerable effort and understanding, probably more than the average student will bring to the problem. Teachers who have experience with other languages will often find the style of Logo rather foreign; the argument in South Australia a year or so ago about stop rules was a consequence of one person's unfamiliarity. There are two ways to overcome these problems: one is to read writers like Harvey, the second is to write in nothing but Logo for a time.

Issues of syntax in Logo are nonexistent once the process of evaluation is understood, again, Harvey is perhaps the best to read. The distinction between " (quote) and : (dots) is easy: " means 'treat this word as just a word', while : means 'return the value bound to the word'. The real intellectual leap is the process of recursion. There are as many ways of understanding that as there are expositors; Harvey gives 4 separate explanations in 4 separate Chapters 5 in the first volume of his trilogy.

Logo as it stands, despite the forthcoming new version of LogoWriter[1], is not the latest, nor necessarily the best, medium for programming in schools. Two paradigms are becoming increasingly important in programming.

The first of these is Object Oriented programming. The 'classic' object oriented language is Kay's Smalltalk, but objects are appearing in versions of other languages, witness C++, Common Lisp Object System, and, you guessed it, Object Pascal (in several forms). (Of course, to use the OOPS features of C++ or Object Pascal, one must first know C or Pascal, which rather defeats the purpose in a learning situation.) There is also Object Logo, a very powerful system for the Macintosh, at present the only truly incrementally compiled Logo. As well, Ada and Modula-2 have some object concepts built into them. Which should be learned first? The consensus seems to be that Smalltalk offers the best grounding, since the concepts are applicable to the other OOPS languages/dialects, and it offers a complete and coherent programming environment, designed, like Logo, for beginning programmers, including children. It was the model for the Macintosh user interface, and the other GUIs. Given that Smalltalk is now available for the MS-DOS and Macintosh environments, a case could be made for abandoning Logo and introducing Smalltalk. (In case that sounds like heresy from a Logophile, look up Smalltalk in *Mindstorms*. Smalltalk has much of the list processing ability of LISP, as well as arrays, and an old friend:

```
Turtle
        newWindow: 'Turtle Graphics'; defaultMib;
                2; darkGray; home;
        3 timesRepeat: [Turtle go: 100; turn: 120]
```

[1] Any Primary school which has not changed to LogoWriter is missing out on a much more intuitive Logo environment, together with some superb teaching materials.

Logos with multiple Turtles already have something of an object flavour: each Turtle with both inherited characteristics and its own.)

The other class of programming (in reality a subset of OOPS) is hypertext. The most visible example is the Macintosh HyperCard, but Guide has been available for MS-DOS machines for some time, and other systems are appearing. For the Apple IIGS there is HyperStudio, still without scripting and a bit fragile, but showing promise. It's possible to use HyperCard without 'programming' in the normal sense at all, but to make the best use some scripting is necessary. HyperTalk is an English-like, interpreted and interactive programming language, and a number of people have found that children take to it easily.

Boxer, the successor to Logo, is hypertext (for 'button' read 'port'), and in the sense that entities are represented by their boxes, object oriented. There are experimental Boxer sites in Australia, the Sunrise School in Melbourne for example.

Pascal has been, and will continue to be, a significant and influential computer language. It was designed for teaching, but not for young students, who need something different, much more immediate and flexible. Pascal is for training, not education. The real issues for students are not the syntax of some particular language, but 'figuring out what we want to compute, how we will decompose problems into manageable parts, and how we will work on the parts.' (Abelson *et al* 1985, p xvi)

References:

Abelson, H. Sussman, G. and Sussman, J. *Structure and Interpretation of Computer Programs*, MIT Press, 1985

Brown, W. Letter to Editor, *Byte*, July 1984, p 16

Harvey, B. *Computer Science Logo Style: Intermediate Programming* MIT Press
This book should be mandatory for anyone teaching with Logo.

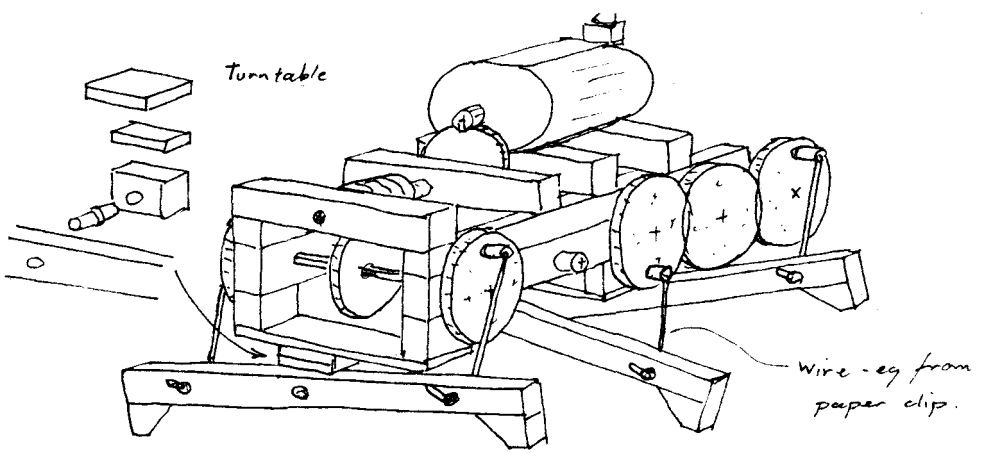
Kay, A 'Microelectronics and the Personal Computer' in *Scientific American*, Vol 237 No 3, September 1977, p 239

Jensen, K. and Wirth, N. *Pascal User Manual and Report* (2nd edition) Springer-Verlag, 1975, p 133

Teller, J. Letter to Editor, *Byte*, December 1984, p 22

More LEGO Legs *(continued from p 6)*

A hexapod remains stable by always having three feet on the ground; the alternating tripod gait. We recently devised another leg mechanism, after deciding that wire was the best means of connecting the gears to the legs, because movement about two planes is necessary. In action it vaguely resembles Sutherland's 'Trojan Cockroach'.



Computing at Entropy House

What is SSABSA Computing Studies about? As people at a recent SACSTA meeting were told, very few students who have done the course are entering computing courses at SAIT or University. Standards of students entering are improving, because of other exposure to computing at school, but one of the problems remaining is a weakness in understanding of abstraction.

In procedural terms,

```
TO Cube :number
OUTPUT :number * :number * :number
END
```

is an abstraction, since it can represent the cube of any number. Which makes more sense when reading a program, IF :number * :number * :number > 100 [do something] or IF Cube :number ... ? Harvey has a good section on data abstraction in Volume 2, pp105..108. Scheming types might work through the index in Abelson, Sussman and Sussman.

Begin Computer Science at Adelaide University in 1990 and learn Ada, the language that has been described as a 'language non-proliferation treaty.' Yes, you can do some LISP, in third year.

Know what 'media ecologists' are? According to Prof Herb Karl (The Australian, October 24th, p 31) they examine media in terms of how they extend human faculties. Of Logo he is reported:

> "Logo is out of favour with educators in the US because it is 'too simple' but that is why it is so great." Professor Karl said.
>
> Kids can learn how to make things happen, develop analytical skills, propositional thinking and see how these processes work in their own minds when using Logo.'

Logo Computer Systems Inc has appointed a new representative in Australia. (Not before time I hear you say.) The company is Computelec Data Systems, of 44 Peninsula Boulevard Seaford Victoria 3198, (03) 786 7177, and they've started off on the right foot by sending fliers about LogoWriter to every secondary school. Ask most secondary, even primary, teachers in South Australia about LogoWriter and you'll get blank looks, so it's good to see some promotion at last. Computelec (the 'e' before the 'l' is sounded) will be handling all LCSI products, as well as the Valiant Turtle.

CEGSA's new monthly newsletter is named *RAMpage*. Perhaps *POALL* needs an heraldic shield, with Turtles RAMpant!