Thompson, P. W. (1986). Logo as a context for thinking about thinking. In R. Noss & C. Hoyles (Eds.), *Proceedings of the Second International Conference on Logo and Mathematics Education*, pp. 209-215. London: London Institute of Education.

Logo as a Context for Thinking about Thinking

Patrick W. Thompson
Department of Mathematics
Illinois State University
Normal, IL 61761 USA

Seven undergraduate students were given the project, completed over a period of two weeks, of developing Logo procedures that would simulate the ways people count.They were not allowed to use numbers anywhere in their procedures: the procedures were required to be entirely "linguistic." The students kept journals of their work and thoughts; excerpts are reported here.

The Course

In Fall of 1983, while at San Diego State University, I taught an experimental course for pre-service elementary school teachers entitled *Algorithms In Elementary Mathematics.*My goal was to focus on algorithmics as a foundation for learning mathematical concepts. The stated goal of the course, at least at its outset, was to teach students to program in Logo. Topics covered included elementary turtle graphics, recursive graphics (fractals), elementary list processing, and recursive list processing. These topics were covered in the context of problems involving analytic geometry, set theory, number theory, and arithmetic.

Seven students participated in the course. Being an experimental course, the students who elected to take it were above average in mathematical background and ability. Also, though it was a course for prospective elementary school teachers, of the seven students there was one prospective secondary teacher, and there were two computer science majors.

There was no text; Abelson's *Logo for the Apple II* was used as a reference. Class assignments were given during class as handouts. Class discussions then focused upon understanding the problem statements and laying out plans of attack. The assignments typically were written so that students had one week to complete them. Each student was required to keep a diary for every assignment; the diary was to contain a travelogue of their work and its relationship to learning and teaching mathematics. The following assignment is an example from number theory. This assignment was to be completed in one week. It was given on October 1, 1983, which was the 6th week of the (15 week) semester.

1. Write a collection of procedures that will generate a list of prime numbers from 2 to an arbitrary upper limit. Store the list as a global value.
2. Write a procedure that takes an arbitrary natural number as input and that outputs that number's prime factorization. The output should be a list containing prime factors of the input; multiple factors should be contained in sublists. Your procedure should output [2 [3 3] [5 5 5] 7] as the prime factorization of 15750.
3. Write a procedure that takes an arbitrary natural number as input and that outputs a list containing all natural divisors of the input (in any order). A number's list of divisors should be generated from its prime factorization.

On this assignment all seven students finished part 1, six finished part 2, and two finished part 3.

The course's focus on algorithmics is shown more explicitly by an assignment on arithmetic. In that assignment students were to model two methods for adding whole numbers: the standard written algorithm ("carrying"), which emphasizes figurative operations upon digits of numerals only, and the more conceptual approach of adding ones, tens, etc. and then converting the sum to standard form. Students found that the former required more "steps" than did the latter but required no numerical knowledge of numeration, while the latter required more relational knowledge about numbers and numeration than did the former.

Modeling Counting

Another project, which is the major topic of this paper, was this:

Write a procedure or procedures that simulate the way a skilled counter counts by ones. You are under two constraints:

I. You cannot use numbers *anywhere* in your procedures.
II. It must be at least plausible that people count the way your collection of procedures counts.

Your procedure must be able to perform as follows:

1. It must be able to count from "one" to "one thousand."
2. It must be able to start with and stop at arbitrary number-names.

The counting project was given on October 13, 1983, which was the 8th week of the semester. Students had two weeks to complete it. They worked in three groups -- two groups of two and one group of three. At the end of the two weeks they were asked to respond to two questions.

1. What did you learn about counting and about how people learn to count?

2. How can you apply what you learned in this project to classroom situations?

Students' journal entries showed that, to a person, they thought the project was going to be quite easy. One student, SK, wrote:

New problem:

> *Teach the computer to count. Use words only. Maybe (certainly) it will be harder than it sounds because it doesn't sound very hard. Couldn't it be done with a list of one through ten and suffixes (teen, ty)?*
>
> *Later - I was right - it's harder than it sounds! A list of one through nine is better than one through ten. What comes after nine changes although it has a pattern too.*

Most students anticipated the repetition of a pattern. They related this insight to a problem they had solved earlier, drawing a regular polygon and connecting every vertex with every other vertex. RM wrote

> ***10/13*** *Assignment on counting .... Well, this will be similar to diagonals [if we use {a, b, c, d, e, f, g, h, i} instead of {one, two, ..., nine}]. Will take 2 lists (the same) and connect the first member of the first list with every member of the second list, then repeat using butfirst of the first list. Slightly different than diagonals. For now, we'll ignore exceptions.*

Another student, MS, wrote

> ***10/16*** *I'm not sure it will work, but I think it will work. I hope so. Had a hard time at first trying to figure out how to have twenty-one, twenty-two, etc. all the way down. Then I remembered connecting the diagonals. I structured my program sort of around that concept.*

As it turns out, similarities between connecting vertices and the way people count are only superficial. The resemblance is more in relationships found within the finished products than in the process by which the products are constructed. Students found this out when they began to consider how to make their procedures begin and end at arbitrary numbers. MS wrote

> ***10/20*** *Tried to count the way I think real children do. Couldn't figure out how to program. How am I going to be able to start anywhere? I need to know what follows each number. How do I do this? Time to go back to working the old way [which was to generate a finished list of number-names and chop those from the front and back that didn't belong] . At least I can get something.*

Several students took an approach similar to MS's, which was to construct a list of number-names and then do with that list what they wished. None who took this approach realized on their own that they were not working within the second constraint listed in their directions: it must be at least plausible that their procedures count the way people count. HM wrote:

> **10/24** *(RM) and I went to Dr. T's office as an afterthought before we started more work on our project. Am I glad we did! Dr. T told us that we had a basic design problem in our program that wasn't following what a real counter would do. We were putting all the numbers [names] in a list and then printing that list. It is so apparent to me now that if someone asked me to count I wouldn't count silently, store the info and then spit out the storage. I would say immediately the number I just thought of. I guess we lost sight of the project's purpose: to pattern what a human being would do when beginning counting.*

All seven students saw that their procedures needed to be recursive -- that the process of counting had a "nested", self-referential structure. However, none saw a need to impose a structure on the number-names themselves. MS's question, given previously, about how one begins counting with an arbitrary number-name, and the succeeding comment about the need to know what name follows any name, was as close as any student came to the realization that names need to be structured just as did procedures that operate upon names. For example, one way to structure "one hundred forty-two" is as [[[ONE] HUNDRED] [[[FOUR] TY] [TWO]]], which is an instance of the general structure [[NAME_1 HUNDRED] NAME_2]. Any reference to a name refers also to the name's structure. This accounts for nonstandard, yet meaningful, constructions such as "thirteen hundred seventy-three."

Students' conceptualizations of the project barred them from modifying their procedures to begin and end with arbitrary number-names. They designed their procedures to produce complete lists. Thus, they had no entry other than at the beginning of a sequence. What they needed was a procedure that takes a number-name as input and which outputs the name coming next after the input. However, this "production system" conceptualization of the problem seemed alien to them.

All students' diaries had entries that showed their introspection on the process of counting. HM wrote:

> **10/25** *We have it counting to "ninety-nine" now. Must be to "one thousand" by Thursday. We're hoping that additions of subprocedures to the main one will accomplish this. .... I feel it must consist of a recursive call back to MPM [their*

> *main procedure] and combining it with the first of T.LIST [ONE TWO ... NINE] and the word HUNDRED. It must be a pattern that can be used for thousands, etc. also.*
>
> *How does a counter count? I wish I knew. Does everyone learn differently? Do they memorize 1-9? 10, 20, 30, 40? I wonder when the counting becomes meaningful instead of some memorized list, that gets praise after it's said.*

Each group developed procedures that would produce sequences from "one" to "one thousand." The strategies were different, yet similar in principle. One group had a major procedure that directed control to sub-procedures for counting from one to ten, eleven to nineteen, twenty to ninety-nine, and one hundred to one thousand. Each subprocedure was given one or two lists of names, along with a suffix to place between parts of a name.

At the time of the project's deadline, no group had procedures that could begin and end with arbitrary number-names. I focused their attention on the issue of representing number names per se, and offered examples like the one given in this paper for "one hundred forty-two." By the next meeting, four days later, two groups had completely revised their procedures so that they behaved as required (see Appendix I).

What Did Students Learn?

Students were uniformly positive about the project, although many admitted frustration because of its difficulty. Some remarks were:

HM *The project made us empathetic with a beginning counter.Even if a teacher tried to teach the computer the names of the United State's states [sic] it would give him/her insights into the thinking of what the students were going to go through .... How much better to do these exercises with a machine!It's better than experimenting with little human minds, at least for an untrained person such as me.*

RM *We have to keep our goal in mind.We must think about <u>how</u> we accomplish a task or arrive at a solution, not just what the outcome or answer is.We can use this method to think about teaching other topics -- long division, for instance (goes on to give specific example of an analysis of long division).....*
The division method is similar to our counting problem in that we should consider "the how." ... It is often difficult to consider the "why" and "how" because many of us never considered them when we learned math.

SK *.... What have I learned in this project?I think I will be more aware of seeing the patterns in the mistakes that students make -- not just in math but in all subjects.*

MS *.... Before we started this project I understood in a vague sort of way how we count. As I tried programs, I knew it wasn't the way we really count, but I couldn't figure out how to make the computer count "correctly."If I'd had a child in front of me, I could have explained to him much better than I could that dumb computer.Now that the pressure is off, I sort of want to go back and try once again.*

It is unclear from the students' diaries as to how deeply they understood the moral of the task I had set for them: that to understand behavior one needs a model of the thinking which is expressed in that behavior. Certainly, they did not understand that it is insufficient to model only procedural knowledge, that one must attend to the structure of the mental "objects" upon which procedures operate. Perhaps this project was too much to ask of undergraduate students. I have hesitated to attempt such a project again, as this group of students was exceptional, and they found it quite difficult. One would think that having prospective teachers analyze nontrivial tasks with the aim of modeling underlying thought processes would be beneficial for them as future diagnosticians of children's errors. However, the level of sophistication this requires may be too great. Nevertheless, one thing is clear: without Logo, this project would not have been feasible even with this exceptional group. Its ability to allow explicit representations of complex structures is essential for developing models of children's thinking.

Appendix I

Logo Procedures for Counting (Translated Into Microsoft Logo for the Macintosh)

```
TO START
   MAKE "BASE.SET  [[ONE] [TWO] [THREE] [FOUR] [FIVE] [SIX] [SEVEN] [EIGHT] [NINE]]
   PRINT [WHERE TO START (SUCH AS: ONE HUNDRED FOUR TEEN)?]
   MAKE "BEGIN DECIPHER READLIST
   PRINT [HOW FAR TO GO (SUCH AS: FIVE HUNDRED THIR TY SEVEN)?]
   MAKE "END DECIPHER READLIST  {Input must be in all caps!!}
   PRINT []
   SAY.SEQ :BEGIN :END
END

TO DECIPHER :X
   IF EMPTYP :X [OUTPUT :X]
   IF SIMPLE?  :X [OUTPUT :X]
   IF ALMOST.SIMPLE?  :X [OUTPUT LIST UNIT FIRST :X LAST :X]
   OUTPUT ( LIST UNIT FIRST :X FIRST BF :X DECIPHER BF BF :X )
END

TO SIMPLE? :X
   OUTPUT EMPTYP BF :X
END

TO ALMOST.SIMPLE?  :X
   OUTPUT EMPTYP BF BF :X
END

TO UNIT :X
   IF LISTP :X [OUTPUT :X]
   OUTPUT ( LIST :X )
END

TO SAY.SEQ :CURRENT.WORD :END
   SAY :CURRENT.WORD
   TYPE CHAR 13
   IF KEYP [HOLD]
   IF :CURRENT.WORD = :END [STOP]
   SAY.SEQ ( NEXT.OF :CURRENT.WORD ) :END
END

TO SAY :X
   IF EMPTYP :X [STOP]
   IF LISTP FIRST :X [SAY FIRST :X TYPE CHAR 32 SAY BF :X STOP]
   TYPE FIRST :X TYPE CHAR 32
   SAY BF :X
END

TO HOLD
   IF KEYP [LOCAL "X MAKE "X READCHAR STOP]
   HOLD
```

```
END

TO NEXT.OF :X
    IF EMPTYP :X [OUTPUT FIRST :BASE.SET]
    IF BASIC?  :X [OUTPUT NEXT.BASIC :X]
    IF TEEN? :X [OUTPUT NEXT.TEEN :X]
    IF TY? :X [OUTPUT NEXT.TY :X]
    IF HUNDRED? :X [OUTPUT NEXT.HUNDRED :X]
    PR SE [I DON'T KNOW WHAT COMES AFTER] :X
    THROW "TOPLEVEL
END

TO BASIC? :X
    OUTPUT MEMBERP :X SE :BASE.SET [[TEN] [ELEVEN] [TWELVE]]
END

TO NEXT.BASIC :X
    IF MEMBERP :X :BASE.SET [OUTPUT NEXT.BASE :X :BASE.SET]
    IF :X = [TEN] [OUTPUT [ELEVEN]]
    IF :X = [ELEVEN] [OUTPUT [TWELVE]]
    IF :X = [TWELVE] [OUTPUT [[THIR] TEEN]]
END

TO NEXT.BASE :X :Y
    IF EMPTYP :Y [OUTPUT [ONE]]
    IF :X = FIRST :Y [IFELSE  EMPTYP BF :Y [OUTPUT [TEN]] [OUTPUT FIRST BF :Y]]
    OUTPUT NEXT.BASE :X BF :Y
END

TO TEEN? :X
    OUTPUT "TEEN = FIRST BF :X
END

TO NEXT.TEEN :X
    IF [NINE] = FIRST :X [OUTPUT [[TWEN] TY]]
    OUTPUT LIST HOM1 NEXT.OF HOM2 FIRST :X "TEEN
END

TO TY? :X
    OUTPUT "TY = FIRST BF :X
END

TO NEXT.TY :X
    LOCAL "Q
    IF ( LAST :X ) = "TY [OUTPUT LPUT [ONE] :X]
    IF ( LAST :X ) = [NINE] [MAKE "Q NEXT.OF HOM2 FIRST :X
    IFELSE  [TEN] = :Q [OUTPUT [[ONE] HUNDRED]][OUTPUT LIST HOM1 :Q "TY]]
    OUTPUT LPUT NEXT.OF LAST :X BL :X
END

TO HUNDRED? :X
    OUTPUT "HUNDRED = FIRST BF :X
END
```

```
TO NEXT.HUNDRED :X
    IF ( LAST :X ) = "HUNDRED [OUTPUT LPUT [ONE] :X]
    IF :X = (LIST FIRST :X "HUNDRED [[NINE] TY [NINE]]) [OUTPUT LIST NEXT.OF FIRST :X
    "HUNDRED]
    OUTPUT LPUT NEXT.OF LAST :X BL :X
END

TO HOM1 :X
    IF :X = [TWO] [OUTPUT [TWEN]]
    IF :X = [THREE] [OUTPUT [THIR]]
    IF :X = [FIVE] [OUTPUT [FIF]]
    OUTPUT :X
END

TO HOM2 :X
    IF :X = [TWEN] [OUTPUT [TWO]]
    IF :X = [THIR] [OUTPUT [THREE]]
    IF :X = [FIF] [OUTPUT [FIVE]]
    OUTPUT :X
END
```