Integrating Mathematics and Logo: A Four-Step Approach

W. George Cathcart

Dr. Cathcart is a professor of education at the University of Alberta. He served on the executive of MCATA and is a former editor of delta-K. *Dr. Cathcart has presented papers at the annual meeting of MCATA and at NCTM general meetings.*

Programming in Logo develops problem solving skills and affects cognitive processes in young children (Clements and Gullo 1984). Having middle school children write Logo procedures that simulate certain mathematical algorithms and processes may also be a worthwhile endeavor.

First, the process sharpens a child's understanding of the mathematical concept or algorithm. Before a procedure can be written, the mathematics are analyzed or broken down into small "mind-sized bites."

Second, programming broadens the child's understanding of Logo and leads to a greater appreciation of the power of Logo. Children discover that Logo can do many things besides drawing designs, geometric figures and graphs.

A Developmental Sequence

Using Logo to enhance a mathematical concept involves four steps. The first two steps focus on the mathematics and reinforce and clarify the mathematics for students; Steps 3 and 4 focus on Logo. To complete these steps, students may need to consult a Logo manual or ask for guidance. The experience will both develop their skill in programming and broaden their understanding of Logo's capabilities.

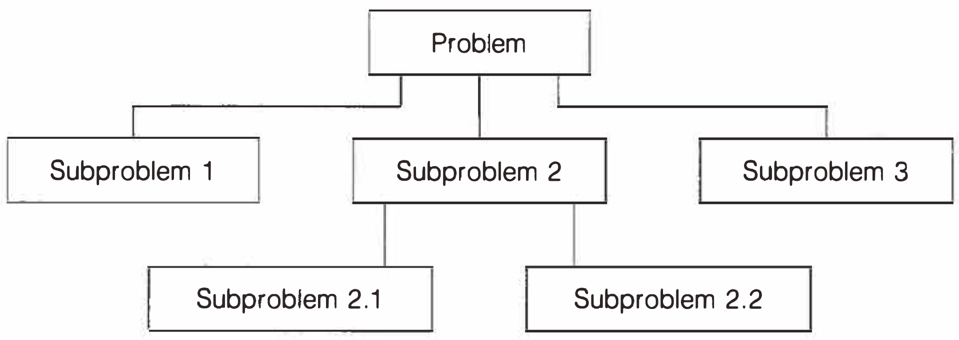
Focus on Mathematics

Step 1. Analyzing the Mathematics

Before students write a Logo procedure to perform a mathematical task, they must be able to break the mathematical process down into its components. Computer scientists call this process "stepwise refinement." Figure 1 illustrates the process.

For example, to find the mean set of numbers, students might analyze the problem by

Figure 1. Stepwise Refinement



1. calculating the sum of all numbers in the list,
2. dividing the sum by the number of elements in the list,
2.1. counting the number of elements in the list, and
3. printing the results.

Figure 2 represents this process using the stepwise refinement chart.

Step 2. Write a Pseudo Code

After analyzing all the components, students should be encouraged to write a pseudo code corresponding to the mathematical steps. Based on the example of the mean, a pseudo code would look something like the following:

```
TO MEAN
  • get sum
  • count number of elements
  • divide sum by the count
  • output the result
END
```

This is just one example of a possible format for a pseudo code. You may prefer a different style. What is important is that students structure their analysis of the mathematical process into a logical step-by-step algorithmic-like statement. The focus here is still on the mathematics. The step is an attempt to state in a succinct form the mathematical process. Additionally, the step serves as a transition to focus on Logo.

Focus on Logo

All the Logo procedures in this article are written in Apple Logo, a product of Logo Computer Systems (LCSI) Inc. It is assumed that the students have a reasonably good understanding of Logo including tail recursion.

Step 3. Logo Match to Pseudo Code

The first attempt at writing a Logo procedure to perform the mathematical task could be a relatively straightforward translation of the pseudo code into Logo. The pseudo code developed in Step 2 can be translated as follows:

```
TO MEAN :ALIST
MAKE "S ADDUP :ALIST
MAKE "C COUNT :ALIST
MAKE "R :S / :C
OUTPUT :R
END
```

Note that ADDUP is used as an operation in line 2. ADDUP, from the context, appears to be an operation that calculates the sum of a list of numbers. ADDUP is not a Logo primitive and should be given to students as a tool procedure, perhaps included in a STARTUP file. There is one version of ADDUP:

```
TO ADDUP :ALIST
IF EMPTYP :ALIST [OUTPUT 0]
OUTPUT SUM FIRST :ALIST ADDUP BUT-
  FIRST :ALIST
END
```

Figure 2. Stepwise Refinement for Finding Means

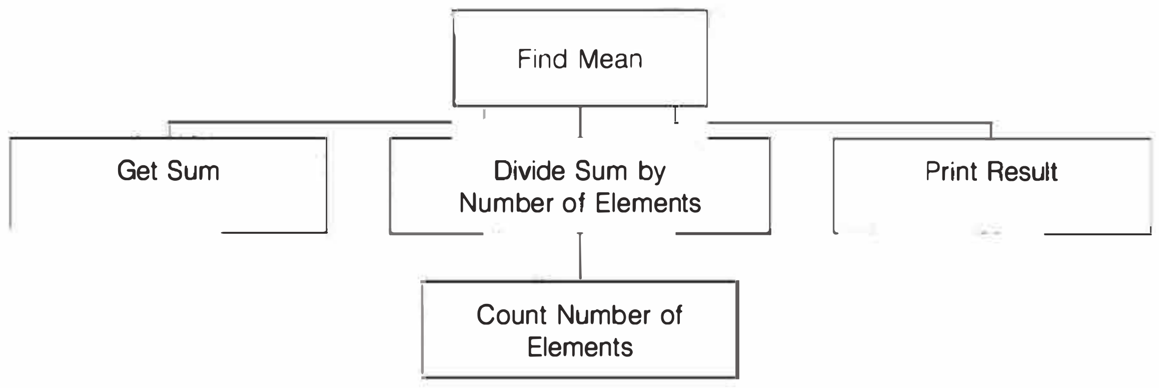


Figure 3. Pseudo Code and Logo Match in Parallel

Step 2. Pseudo

```
TO MEAN
  • get sum
  • count number of elements
  • divide the sum by the count
  • output the result
END
```

Step 3. Logo Match

```
TO MEAN :ALIST
MAKE "S ADDUP :ALIST
MAKE "C COUNT :ALIST
MAKE "R :S / :C
OUTPUT :R
END
```

A worthwhile exercise for students is to do Steps 2 and 3 side-by-side on a page (see Figure 3).

The pseudo code and the Logo match may not always correspond as closely as in Figure 3. Minor adjustments to format may be required for some tasks. However, if the pseudo code is a detailed statement of the mathematical process, Logo statements can usually be written to match it step-by-step.

Logo "purists" will be dismayed by the extensive use of global variables in this approach. The situation was a trade-off, I wanted a strategy that would make the transition from mathematics to Logo a small step. Steps 2 and 3 seem to do this. I am prepared to trade some purism for simplicity. Step 4 restores some of the "pure" Logo.

Step 4. More Elegant Logo

In Step 4, encourage students to ask "Can I write this procedure in a better way?" The purpose is to write a more elegant procedure. Students should consider shortening statements, combining statements, using different commands or operations, taking a completely different approach and so on.

A more elegant procedure for our example of the mean might combine all of the steps into one line:

```
TO MEAN "ALIST
OUTPUT (ADDUP :ALIST) / COUNT :ALIST
END
```

In writing a more elegant Logo procedure, students should be encouraged to eliminate as many MAKE statements as possible. (Note the removal of all MAKE statements in the final MEAN procedure.) After students work through this four-step process with different mathematical concepts, you may wish to explain the difference between local and global variables in Logo and the advantages and disadvantages of each.

With some experience, students may translate the pseudo code (Step 2) into a Logo procedure that is somewhat "better" than a straight line-by-line translation. However, Step 4 still needs to be emphasized because a search for a more elegant procedure should never end. (To illustrate this point, refer to the example of the median in the following section.)

Students soon discover that the four-step process does not always result in a Logo procedure that will do exactly what was expected. A bug may have crept in during the transition from pseudo code to Logo. The pseudo code itself may contain errors, such as missing steps. Possibly the analysis of the mathematics was incorrect. Therefore, a continual evaluation or monitoring of each step and the total process must be practiced. Figure 4 illustrates the process of monitoring.

More Examples

To illustrate the outlined four-step process of integrating Logo into mathematics, two additional nongraphic examples are outlined. One deals with divisors (factors), the other with the median.

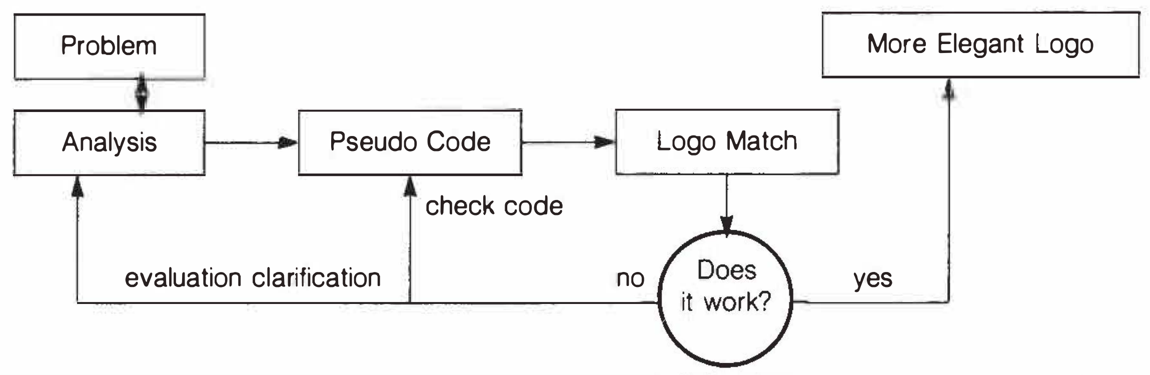
Factors

The problem is to find all the factors of a whole number, n.

Step 1. Analysis

A factor of n is a whole number that divides evenly into n. One method of obtaining all the divisors of a number is to check for divisibility (remainder = 0) by all whole numbers less than or equal to n. If divisibility occurs, list the divisor as a factor; otherwise try the next whole number. Except for n itself,

Figure 4. Dynamics of the Four-Step Process



check divisors of whole numbers up to n/2 because no number between n/2 and n will divide evenly into n. A child thinking through the concept of a factor in this way clarifies the concept in his or her mind.

Step 2. Pseudo Code

There are several ways the analysis could be translated into succinct logical steps. Here is one possibility:

```
TO FACTORS of n
  • start with divisor of 1
  • if divisor > n/2, print n as a factor and stop
  • otherwise get remainder when n/divisor
  • if = 0, list divisor as a factor
  • otherwise, repeat with divisor 1 greater
END
```

Step 3. Logo Match

```
TO FACTORS :NUM :DIV
IF :DIV > :NUM / 2 [PRINT :NUM STOP]
MAKE "R REMAINDER :NUM :DIV
IF :R = 0 [PRINT :DIV]
FACTORS :NUM :DIV + 1
END
```

The pseudo code and the Logo match do not correspond as closely as in the example of the mean described earlier. The initial divisor, 1 (line 2 of the pseudo code), is incorporated as the second input to the procedure. A sample execution of FACTORS might be: FACTORS 36 1. Different versions of pseudo code may result in a greater or lesser degree of correspondence in the Logo match.

Step 4. More Elegant Logo

The first improvement would be to combine lines 3 and 4 and remove the MAKE command:

```
TO FACTORS :NUM :DIV (assign :DIV the
  value 1)
IF :DIV > :NUM / 2[PRINT :NUM STOP]
IF REMAINDER :NUM :DIV = 0 [PRINT :DIV]
FACTORS :NUM :DIV + 1
END
```

This version places the factors in a vertical format. A more elegant procedure should print them horizontally with a space or a comma between each factor. Changing the THEN portion of line 3 to [(TYPE :DIV CHAR 32)] would cause this to happen.

As it stands, FACTORS is rather limited in its usefulness. To be used in another set of procedures, say for finding common factors, FACTORS needs to be an operation or outputting procedure. Unfortunately, PRINT or TYPE cannot simply be replaced by OUTPUT since OUTPUT also stops the procedure. This is a case in which students will likely need your assistance.

One way around the dilemma is to store each factor in a list, and output the list when all the factors have been determined. The following procedure, while longer, is more elegant; results can be used as input to other procedures in which factors are needed.

```
TO FACTORS :NUM
MAKE "FACTS [ ]
GET.FACTORS :NUM 1
```

```
OUTPUT SENTENCE :FACTS :NUM
END
TO GET.FACTORS :NUM :DIV
IF REMAINDER :NUM :DIV = 0 [MAKE
  "FACTS LPUT :DIV :FACTS]
IF :DIV < :NUM / 2[GET.FACTORS :NUM
  :DIV + 1]
END
```

Sample Output

```
?PRINT FACTORS 16
1 2 4 8 16
```

LPUT is the operation storing each factor as it is generated into the list, FACTS. Notice that, since a list was created, the output is automatically in horizontal form.

Median

I dealt with the concept of the median in a previous issue of *delta-K* (Cathcart 1986). Some of the ideas presented in that article can be incorporated into the four-step process.

Step 1. Analysis

For the median to be found, the data needs to be sorted. The number of elements in the data list must be determined. If this number is odd, the middle number is the median. Otherwise, the median is the average of the middle two numbers.

Step 2. Pseudo Code

```
TO MEDIAN
  • sort input list
  • count number of elements in input list
  • check if number of elements is odd or even
    • if odd, pick middle element
    • otherwise find mean of middle two numbers
  • output result
END
```

Step 3. Logo Match

```
TO MEDIAN :ALIST
MAKE "SL SORT :ALIST
MAKE "C COUNT :ALIST
TEST (REMAINDER :C 2) = 0
IFTRUE [EVEN]
IFFALSE [ODD]
OUTPUT :R
END
```

This procedure calls two subprocedures, EVEN and ODD. SORT is a tool procedure that sorts data into ascending order. This procedure should be given to students as a tool. (See Cathcart 1986, for a listing of a sort procedure.)

```
TO EVEN
MAKE "F ITEM :C / 2 :SL
MAKE "K ITEM :C / 2 + 1 :SL
MAKE "R MEAN LIST :F :K
END
TO ODD
MAKE "R ITEM :C / 2 + .5 :SL
END
```

Step 4. More Elegant Logo

A first attempt to write a more elegant procedure may result in the following:

```
TO MEDIAN :ALIST
MAKE "C COUNT :ALIST
IF (REMAINDER :C 2) = 0 [OP EVEN] [OP
  ODD]
END
TO EVEN
OP MEAN LIST ITEM :C / 2 :ALIST ITEM :C
  / 2 + 1 :ALIST
END
TO ODD
OP ITEM :C / 2 + .5 :ALIST
END
```

With these procedures, SORT would be used as an input to MEDIAN. That is, PRINT MEDIAN SORT :ALIST. This example shows how trying to modify a procedure, while retaining the basic strategy, may blind a programmer to a far more elegant solution. By determining the median with pencil and paper, it is possible to simply strike out the first and last elements of the sorted data. Continue this process until only one or two elements are left. If one element remains, it is the median. If two elements remain, the average of these two is the median. Actually, if only one element remains, the median is still the average of the number. To illustrate:

Case 1: ~~1~~ ~~3~~ ~~9~~ 16 ~~25~~ ~~31~~ ~~33~~
16 is the median

Case 2: ~~1~~ ~~3~~ 9 $\downarrow$ 16 ~~25~~ ~~31~~
12.5 is the median

This suggests a recursive procedure. A much more elegant procedure for calculating the median, then, would be as follows:

```
TO MEDIAN :ALIST
IF OR ((COUNT :ALIST) = 1) ((COUNT :ALIST)
   = 2) [OP MEAN :ALIST] [OP MEDIAN BF BL
   :ALIST]
END
```

Line 2 (IF–THEN portion): if there are one or two elements in the data list, output the mean and stop.

Line 2 (ELSE portion): if there are more than two elements, strip away the first and last and repeat the process.

Summary

A four-step process for integrating Logo programming into mathematics consists of

1. analyzing the mathematics,
2. writing a pseudo code for the mathematical process,
3. writing a Logo code to correspond to the pseudo code, and
4. writing a more elegant Logo procedure.

Some steps may be repeated or revised as the process develops. This action may be needed to correct the code or to re-analyze the problem due to initial misconceptions or omissions.

If students follow the four steps, they will likely sharpen their understanding of the mathematics involved, broaden their knowledge of Logo, increase their appreciation of Logo and hone their programming skills.

References

Cathcart, W.G. "Logo and Measures of Central Tendency." *delta-K* 25, no. 3 (1986): 27–31.

Clements, D.H., and D.F. Gullo. "Effects of Computer Programming on Young Children's Cognition. *Journal of Educational Psychology* 76, no. 6 (1984): 1051–58.