Homework 1: Pseudo-Code Interpreters

This homework is due Thursday, Jan. 26 at the beginning of your recitation section.

1. In the design of our pseudo-code (see chapter 1 of the text) there is no "inverse operation" that corresponds to the move operation in the way that subtraction corresponds to addition or that division corresponds to multiplication.

   (a) (2 points.) What function, that is not easily or efficiently accomplished with the other operations should be added to our pseudo-code (with operation code -0) as the "inverse operation" for move?

   (b) (4 points.) Justify your answer by describing: why the function you chose above cannot be easily or efficiently accomplished with the other operations and why this function satisfies the regularity principle.

2. (2 points) Write a complete pseudo-code program to read in data cards (until a +0000000000 flag card), add up the numbers on the cards, and print out the sum.

3. (10 points) Write an interpreter for the pseudo-code described in section 1.2 of the text, except that all numbers are integers. (Truncate the results of the division and square root operations.) The interpreter should be written in Pascal. Do not implement your own integer arithmetic; just use the integers of Pascal. You do not have to include the operation you designed in the first problem above. Mail the source for your program to the teaching assistant (matthews), and turn in print-out of your program's source, and results of a test on the example of figure 1.3 in the text (with input). (This program and some input will be available next week on Zaphod in the file `/users/class/cs342/public/pseudo-code.ex`.)

4. The pseudo-code interpreter designed in section 1.3 has some cost compared to direct programming. The overhead in time can be estimated by counting memory references (fetches or stores to memory).

   (a) (2 points.) Estimate the number of memory references made in the read-execute cycle of an interpreter, beyond those actually required for computing the result (e.g., a floating-point operation).

   (b) (2 points.) What percentage of memory references will be overhead if the average software-implemented floating-point operation requires 100 memory references?

   (c) (2 points.) What percentage of memory references will be overhead if floating-point operations are implemented in hardware and take just three (3) memory references?