Hello, students!

I would like you to do *one* programming assignment, which should be fun. Email me your answer by next Sunday midnight, 26 July. gries@cs.cornell.edu. You have to write *one* loop. I had to write this loop myself a week ago, and I want to see how well *you* do with it.

Visit the website http://2048game.com and play the game 2048. (Google 2048 to find other websites for playing the game.) It's played on a 4 x 4 board. You move numbers around on the board using the arrow keys: up, left, right, and down. Play the game for a while, so that you fully understand it. But don't play for too long! It is addictive.

Now, consider clicking the left arrow, so that a move is made to the left. I want you to write part of this move. Below, I give the specification of a Java procedure to write, and below that, I give some rules that the move must follow.

The values are kept in square array b. So moving left consists of moving elements of each row b[i]. We give examples below. Please note that the game is played on a 4 x 4 board, but a larger board could be used — 5 x5 or 6 x 6. So in writing the method, do not assume that b is a 4 x 4 array. It is a square array of size b[b.length] [b.length], and you have to write a loop to do the move.

Note: If you don't know Java, do it in some other language; you may have to give the length of the array as a parameter in that case.

```
/** Perform a move in direction left on row b[i].
  * Return the points gained for this move (-1 if no move made). */
public int moveLeft(int[][] b, int i) {

}
```

The following examples illustrate the rules to be moved. Remember that 0 denotes an empty cell.

1. Values get moved as far to the left as possible. (0, 8, 0, 2) becomes (8, 2, 0, 0).

2. Two neighboring equal values get added and stored as one value. This is the only time points get produced. The points produced is that one value. (4, 4, 16, 16) becomes (8, 32, 0, 0). The number of points is 8 + 32 = 40.

3. Once a merged value has been produced, it cannot participate in another merge (during this move). Thus, (4, 4, 0, 4) becomes (8, 4, 0, 0), with a point score of 8. Note that the *leftmost* two 4's are added together. It is *wrong* for this move to produce (4, 8, 0, 0). Also, (4, 4, 4, 4) would become (8, 8, 0, 0).

4. Based on the above rules, it makes sense to process the array from beginning to end.

My encounter with this problem and how I solved it.

I found a solution to the 2048 game, which played it automatically using either a minimax algorithm or alpha-beta pruning, and we are considering using part of it as an assignment this fall in the second programming course, *OO Programming and Data Structures*. However, as you can imagine, I had to rewrite parts of the program completely, with method specifications, good comments, including loop invariants, etc. The structure had to be changed somewhat, and I built a GUI for it in Java.

I could not understand the original code in method moveLeft! So I rewrote it completely. It needed a loop, and I figured out what the loop invariant was, wrote the loop and the rest of the method body, and it worked the first time except for one typo that I made.

I did not develop the loop invariant formally by combining pre- and post-conditions, or exactly by any other method we talked about in class. But I *did* write it carefully before writing any code. For example, one part was:

b[i][0..h-1] has its final value and cannot be changed again

Do your best with this. Try to come up with a suitable loop invariant, and if you do, you will be surprised a how easy it is to write the algorithm. I'll make my solution after I have seen all or yours.

It's also OK to test your method if you have an available IDE around to help you. I suggest you do that.