Name partner 1: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name partner 2: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

How do we solve programming problems?

Example with Sample Assessment 2\_7

In this class, we are (and will) spend most of our time solving complex problems using Java, the programming language of our choice. However, the hard part of these assignments is not Java itself but breaking down the problem into steps a computer can understand. What we’re really doing is translating a human solution into a computer solution. We’re going to practice that today.

Outline of how to solve programming problems in this class.

1. Understand the problem by looking at test cases
	1. What are the use cases? What are some things we need to consider?
2. What is the human solution? Blow up the problem!
	1. Can we make diagrams to explain our solution?
3. What is in our toolbox? How can we use what we’ve learned in class?
4. Translate the human solution, using what’s in our toolbox, into pseudocode
5. Translate the pseudocode into Java code.
6. Run the tests, and see if there are patterns in what is failing
	1. If there are patterns, you were missing a conceptual step
	2. If there aren’t patterns, revisit your human solution; did we consider all of the test cases?

Let’s go through these steps in order on one of the more difficult problems, Sample Assessment 2\_7 😊

1. **The Mirror Problem (copied from the website here for convenience)**

Imagine that the user specifies with width and height of a grid and provides a tile in that grid. You will write code to return the mirror image of that tile if the grid were folded along its horizontal axis. For example, on a 7x10 grid,

tile 23 will return tile 44, and visa-versa.

If the height of the board is odd, and the tile falls along the horizontal axis, a tile should return itself as its mirror image.

This board has a height of 10, which means its horizontal axis lies between tiles 29 and 36.

Now, what if the board had a height of 11 instead of 10?

This would mean the horizontal axis would be tiles 36 through 42. What does that mean in terms of mirrors of tiles along that row?

Well, let’s take a look at the test cases to finish understanding the problem!

1. **Understand the problem by looking at test cases**

Here are the test cases provided with this problem. Let’s take a look at the ones for boards with odd heights, highlighted in bold:

System.out.println("test1: " + (makeMirror(1,1,1)==(1)));

 System.out.println("test2: " + (makeMirror(1,2,1)==(2)));

 System.out.println("test3: " + (makeMirror(1,2,2)==(1)));

 System.out.println("test4: " + (makeMirror(2,1,1)==(1)));

 System.out.println("test5: " + (makeMirror(2,1,2)==(2)));

 System.out.println("test6: " + (makeMirror(2,2,1)==(3)));

 System.out.println("test7: " + (makeMirror(2,2,2)==(4)));

 System.out.println("test8: " + (makeMirror(2,2,3)==(1)));

 System.out.println("test9: " + (makeMirror(2,2,4)==(2)));

 System.out.println("test10: " + (makeMirror(2,3,1)==(5)));

 System.out.println("test11: " + (makeMirror(2,3,2)==(6)));

 System.out.println("test12: " + (makeMirror(2,3,3)==(3)));

 System.out.println("test13: " + (makeMirror(2,3,4)==(4)));

 System.out.println("test14: " + (makeMirror(2,3,5)==(1)));

 System.out.println("test15: " + (makeMirror(2,3,6)==(2)));

 System.out.println("test16: " + (makeMirror(3,2,1)==(4)));

 System.out.println("test17: " + (makeMirror(3,2,2)==(5)));

 System.out.println("test18: " + (makeMirror(3,2,3)==(6)));

 System.out.println("test19: " + (makeMirror(3,2,4)==(1)));

 System.out.println("test20: " + (makeMirror(3,2,5)==(2)));

 System.out.println("test21: " + (makeMirror(3,2,6)==(3)));

 **System.out.println("test22: " + (makeMirror(3,3,1)==(7)));**

 **System.out.println("test23: " + (makeMirror(3,3,2)==(8)));**

 **System.out.println("test24: " + (makeMirror(3,3,3)==(9)));**

 **System.out.println("test25: " + (makeMirror(3,3,4)==(4)));**

 **System.out.println("test26: " + (makeMirror(3,3,5)==(5)));**

 **System.out.println("test27: " + (makeMirror(3,3,6)==(6)));**

 **System.out.println("test28: " + (makeMirror(3,3,7)==(1)));**

 **System.out.println("test29: " + (makeMirror(3,3,8)==(2)));**

 **System.out.println("test30: " + (makeMirror(3,3,9)==(3)));**

 System.out.println("test31: " + (makeMirror(7,10,23)==(44)));

 System.out.println("test32: " + (makeMirror(7,10,44)==(23)));

Can you draw a 3x3 board below, and then explain what the mirror tiles are of the tiles of the middle row, according to the test cases? Answer below:

So it seems like we have two use cases to possibly deal with: 1) boards with even height and 2) boards with odd height.

Okay, it seems like now we understand how our code is supposed to behave for all possible board shapes and sizes.

1. **What is the human solution? Blow up the problem!**

Now, we need to get inspired how to solve this problem. First, we’ll start small and see if we can figure out how humans might do this on an example that’s easy to visualize. Then, we’ll move to imagining a very large grid that we can’t just “look at” to see the right answer, to help our brains think algorithmically.

Flip to the last page of this handout and rip off the last sheet that contains the example from before. Feel free to flip, fold, cut, measure, and draw on that sheet of paper – how would you calculate the mirror for tile 19?

When you’re ready, list some steps/ideas you had below from physically playing with the grid on the last sheet:

Okay, great! Now we have some inspiration for how to solve this problem in general.

While it was useful to try to get some part of the solution for a smaller example we feel comfortable with, we need our code to work for all possible boards. As a next step, go ahead and rip off the second to last sheet of paper from this handout – it is blank.

Next, pretend that grid is an 807 by 1103 sized grid – it’s huge! Try to draw the tile numbers in the top left, top right, bottom left, and bottom right corner. You should have something that looks like (but yours will be mostly blank instead of the …):

|  |  |  |
| --- | --- | --- |
| 1, 2, 3, … | … | …, 805, 806, 807 |
| 808, … | … | …, 1614 |
| … | … | … |
| … | … | …, some huge number |

Label the width and height of the board on your sheet as well.

Now, if I had a million dollars, and gave you a weekend to solve this problem, could you tell me what the mirror is of tile 4655 on this board? [I don’t have a million dollars to give out, sorry, I wish I did…]

Yes! In the worst case, someone well-motivated could tape together enough graphing paper to draw out the board above, label all the tiles, and then…what? How would you calculate the mirror of that tile?

You really want that million dollars, and I bet you could come up with the correct solution! How would you do it? Can you list some of the steps and/or formulas below?

What we’ve just done is come up with a general, human solution to this problem by blowing up the problem to a test case so large we can’t just take a look at the grid to know the correct answer: we are forced to calculate something.

That was the hardest and trickiest part of problem solving…but it’s also not necessarily the worst, if we always think back to how a human would solve this problem. You’re answer might be different than mine, but here is what I had:

1. Calculate the row and column of the tile
2. See if the board is even or odd in height.
3. If the board is even in height,
	1. Subtract the row from the height of the board to get the mirrorRow
	2. The mirrorCol is equal to the column
	3. Calculate the mirror tile using the mirrorRow and mirrorCol
4. If the board height is odd, the steps are the same as in (3)

Did your solution handle both even and odd board heights? If not, don’t worry – the test cases would have likely failed for the set that you were missing, and you would have known to go back and add in some logic. If you haven’t already, edit your notes to include calculations for both types of boards. Note that if your solution was similar to mine, I happened to have one that worked for both types of boards, but there are other equally valid solutions where you would need to handle these two cases separately.

Now, we need to translate these ideas into code. To do that, let’s see what’s in our toolbox.

1. **What is in our toolbox?**

We’ve learned some basic, but extremely powerful syntax this semester. Let’s take a look at what we know how to do (on the left) and what we have in memory for this problem:

|  |  |
| --- | --- |
| Mathematical operators: + - \* / % | width |
| Variable assignment with = | height |
| if-else statements, along with> < >= <= == != ! | tile  |
|  | row = (tile – 1) / width |
|  | col = (tile – 1 ) % width |
|  | constants (like 1, 2, 3, etc.) |

We’ll also be adding to this toolbox some other conceptual calculations like row and col above. You may have seen some of these in the other exercises (if not, you will):

|  |
| --- |
| Adding/subtracting 1 from the tile to gets its left/right neighbor |
| Adding/subtracting the width from the tile to get its top/bottom neighbor |
| Checking if the tile is on an edge by comparing its row/col with something |
| Checking if something is even or odd using modulus |
| Convert a row and column into a tile number and visa-versa |
| … |

Now, all we need to do is translate our human solution into pseudocode (using what’s in our toolbox above), and then finally polish it into Java 😊

1. **Translate the human solution, using what’s in our toolbox, into pseudocode**

Go ahead and try to translate the solution you came up with earlier into pseudocode, making sure that you’re only using what’s in your toolbox. In particular, this will mean translating expressions to refer to the variables in memory, and/or creating new variables from the existing variables. Write your answer below:

Let’s take a look at how I might have translated my original solution using the toolbox:

1. Calculate the row and column of the tile
2. See if the board is even or odd in height.
3. If the board is even in height,
	1. Subtract the row from the height of the board to get the mirrorRow
	2. The mirrorCol is equal to the column
	3. Calculate the mirror tile using the mirrorRow and mirrorCol
4. If the board height is odd, the steps are the same as in (3)

Could become

1. Calculate the row and column of the tile

row = (tile – 1) / width

col = (tile – 1) % width

1. See if the board is even or odd in height

if (height % 2 == 0) or if (height % 2 == 1)

1. If the board is even in height,
	1. Subtract the row from the height of the board to get the mirrorRow

mirrorRow = height - row

* 1. The mirrorCol is equal to the column

mirrorCol = col

* 1. Calculate the mirror tile using the mirrorRow and mirrorCol

mirrorTile = mirrorRow \* width + mirrorCol

1. If the board height is odd, the steps are the same as in (3)
2. **Translate the pseudocode into Java code.**

This part is pretty straightforward, and the compiler will tell you if there are any mistakes:

public static int makeMirror(int width, int height, int tile){

int row = (tile – 1) / width;

int col = (tile – 1) % width;

//skipping checking if the board is even or odd in height because it didn’t make a difference in my solution

int mirrorRow = height – row;

int mirrorCol = col;

int mirrorTile = mirrorRow \* width + mirrorCol;

 return "" + mirrorTile; //converts the result to a String

 }

1. **Run the tests, and see if there are patterns in what is failing**

Does your code pass all the tests?

If not, can you identify a pattern in the tests that it is failing? This usually because you’re missing a use case, like handing odd-height boards.

If there are no patterns to the failing tests, it might be time to revisit the drawing board and reconsider how the human solved this problem – is there an easier way to do it, focusing on what is in our toolkit?

