<u>COP 3502: Program #3 - Suggested Functional Breakdown</u>

Program 3 can be broken down into 7 separate smaller programs. If you are having trouble with it, I suggest you write these programs separately and then work on figuring out how to turn the programs into functions and stitching together the appropriate functions calls to solve the whole program.

<u>Problem 1: Given the correct color combination and a guess, determine the number of black pegs the guess should receive.</u>

For example, if the correct combination is

```
0    2    1    1
```

and the guess is

```
2    0    3    1
```

then there is exactly one color in the correct spot, so this guess should receive 1 peg. Write a program that correctly solves this problem for any correct color combination and any guess.

<u>Problem 2: Given a color combination stored as each color in order, store it in a frequency table that just stores how many times each color is in the combination</u>

Note that in this conversion, we lose information. Namely, if we are given the frequency information, it's generally impossible to figure out the original color combination that it came from because there are typically multiple color combinations that map to the same frequency information.

For example, if a combination is

```
0    2    1    1    2    3
```

Then we can store the frequency information (assuming there are 4 possible colors) as follows:

index	0	1	2	3
freq	1	2	2	1

<u>Problem 3: Given the frequency tables of the correct color combination and a guess, determine the number of total pegs (both black and white) the guess should receive</u>

For example, let the correct color combination have the frequency table shown above. Consider the following frequency table for a guess:

index	0	1	2	3
freq	0	3	1	2

In this case, there are two 1s in each combination, one 2, and one 3, for a total of four matches.

Note that as previously discussed this data is insufficient to determine the exact number of correct or incorrect matches. It can only be used to determine the total number of matches (sum of correct and incorrect ones.)

Problem 4: Given a correct color combination and a guess, determine how many white pegs the guess should receive

This can very, very easily be solved based on the answers to problems 2 and 3. It's the one piece in this breakdown that probably doesn't deserve its own function in the bigger programs.

Problem 5: Given a color combination as well as a single guess and the response it got, determine if it's possible or not for that color combination to be the correct one.

If we are given a color combination, say

```
0    2    1    1    2    3
```

and the guess

```
1    1    1    2    3    2
```

And we receive 2 black pegs and 2 white pegs, we can ascertain that the color combination shown first CAN NOT be the correct one. This is because had it been correct, the feedback we would have received for the guess shown would have been 1 black peg and 3 white pegs.

Problem 6: Given a color combination as well as a set of guesses and the responses they got, determine if it's possible or not for that color combination to be the correct one.

Some may see this as being coded in the same function as program #5, which is perfectly fine. Basically, once program #5 works, all we need to do is check to see if the color combination is consistent with ALL of the clues individually. (Basically, for each clue, we must find that our color combination is a possible one that gives the exact same feedback that was received for each guess.)

Problem 7: Generate all possible color combinations (done recursively)

If we solve problem #6, then we can check for a single color combination, if it's possibly the correct one based on the clues we have. So, if we have a mechanism for generating ALL of the possible color combinations, all we have to do is try each color combination and add 1 for each one that is possible, based on the clues.