

**2012**

Solutions and Notes Packet

|  |  |  |
| --- | --- | --- |
| **Problem** | **Point Value** | **Number of Correct Solutions** |
| Problem 1: Piggy Bank | 2 | 34 |
| Problem 2: Who Sells More Newspapers? | 3 | 36 |
| Problem 3: Numerical Order | 3 | 15 |
| Problem 4: Hidden Message | 4 | 26 |
| Problem 5: Loan Amortization | 5 | 8 |
| Problem 6: Palindromes | 6 | 24 |
| Problem 7: Grade the Scantron | 6 | 19 |
| Problem 8: Around the Block | 7 | 2 |
| Problem 9: Almost the Same | 8 | 2 |
| Problem 10: Factorials | 9 | 17 |
| Problem 11: Let’s Buy Land | 10 | 1 |
| Problem 12: Crack the Maze | 11 | 5 |
| Problem 13: Score the Test | 12 | 0 |
| Problem 14: Shortest Flight Path | 13 | 1 |
| Problem 15: Pattern Part Play | 25 | 0 |
| Problem 16: Bizarro World Rover | 35 | 0 |
| Problem 17: Pushing Blocks | 30 | 0 |
| Problem 18: Stuck-At Faults | 25 | 1 |
| **Total Possible Points** | **214** |  |

**Problem 1: Piggy Bank**

This problem was pretty straightforward. Count the number of pennies, nickels, dimes, quarters, and half dollars. Ignore everything else, but duplicates are okay. Most teams that got an incorrect solution on this problem submitted a solution that did not print out two decimal places when the number of cents in the answer was a multiple of 10 (which was the case with the judging input file), so we saw a lot of $260.3 answers.

Your first instinct was probably to use a floating point number (float or double) to keep track of the value because you are thinking in dollars, but then you have to mess with formatting. If you compute the total value in pennies, you can use an integer data type and use division and mod (/ and %) to get the dollars and cents without having to worry about output formatting.

**Problem 2: Who Sells More Newspapers?**

Most of the questions about this problem revolved around the math. Here’s how it works out:

There are N houses on each side of the block. The input file gives you the number of Times subscribers – call that X1. So, for the one side of the block:

Times = X1

Herald = N – X1

On the other side of the block, we know the number of herald subscribers – call that X2. So, for that side of the block:

Times = N – X2

Herald = X2

So the total subscribership for both sides of the block is this:

Times = N + X1 – X2, or N + (X1 – X2)

Herald = N – X1 + X2, or N – (X1 – X2)

It’s the difference between X1 and X2 that shows us the total difference in subscribers. Each newspaper is offset by this difference – Times has it added to N, and Herald has it subtracted from N. So, the total difference between the newspapers is:

Times – Herald = [N + (X1 – X2)] – [N – (X1 – X2)], which equals 2(X1 – X2).

If 2(X1 – X2) is zero, then they have equal subscribers. If it’s positive, then Times has more. If it’s negative, then Herald has more.

**Problem 3: Numerical Order**

This was another pretty straightforward problem. I don’t remember any questions being asked about this one, so I think most teams thought it was clear enough.

**Problem 4: Hidden Message**

This one was also fairly simple. The only question about this one was whether line breaks in the input file should translate to the output, which they should.

**Problem 5: Loan Amortization**

This one was a tiny bit tricky, so we wound up sending out a hint with about 30 minutes left in the contest since so many teams attempted this but none had solved it correctly prior to sending out the hint. The judging inputs contained the two boundary conditions – one where zero payments had been made, and the other where all payments had been made. The second condition results in a tiny negative number (less than a penny) due to rounding, so you have to put a check in your code that sets the remaining principal to 0 if it is negative. Once we sent out the hint about the loan being paid off and -0.00 not being a solution, we had a bunch of correct submissions all at once.

**Problem 6: Palindromes**

This problem gave some teams fits. The key here is the removal of all non alphanumeric characters. Most often, I saw teams using the “\w” character class to remove special characters. The problem with this is that “\w” will let an underscore through, and not replace it.

After the special characters are filtered, it’s just a matter of reversing and comparing the string. Knowing the Java API well will result in a very short implementation, although it’s definitely possible to do a more brute-force method using loops.

**Problem 7: Grade the Scantron**

Most of the questions about this problem were of the “are there always 5 columns of answers?” variety. That suggested that the majority of teams asking questions were splitting up the answer rows into columns, which was (in this particular problem) an unnecessary step.

The other question we received frequently was about rounding the answer up or down if the number of points per problem happened to contain a decimal. The response was always that the points per problem would be an integer.

**Problem 8: Around the Block**

Another fairly straightforward problem, although there is some work to do in computing side lengths and splitting up the phrase into sections. I don’t believe there were any questions on this problem.

**Problem 9: Almost the Same**

The most frequent issue that students had with this problem was with capitalized words. The problem statement had these two sentences in it:

“Capitalization should be ignored when comparing if the words are one letter different or not, but should be retained for alphabetization purposes. In the event that there are duplicate words, the first instance of the word should be kept and the rest of the instances discarded.”

This means that teams would need to keep a record of used words ignoring case, but use the actual word (capitalization included) when printing out the word after doing the comparison.

A good example of this in the judging input file is the word “till”. The first instance is at the beginning of a line, so it should be “Till” in the output, but frequently we saw either “till” by itself, or both “Till” and “till” in teams’ outputs.

**Problem 10: Factorials**

This was probably the most incorrectly solved problem. Teams that used a primitive data type (like int or long) got the answer wrong because of overflow errors. Teams needed to use the BigInteger class which lets you work on arbitrarily large integers (constrained by your computer’s memory, of course).

**Problem 11: Let’s Buy Land**

This was the first of two exhaustive search problems. It’s not extremely difficult, but there are a ton of nested loops. First, you have to loop through all possible plot sizes. Then, for each plot size, you have to loop through all possible positions of a plot that size. Then you have to look through the value data and add up the values for the plot that you’re looking at. Once all that is done, then you can compare to the best case you’ve found so far. The only question I remember about this problem had to do with the photocopy – the shaded area was very light on the copied version so one team asked what the shaded area was just to double check it.

**Problem 12: Crack the Maze**

Based on the questions received about this problem, the only unclear thing was where the path would start and end. There were lots of questions about if the path could lead to the top or bottom, and whether that would be considered an exit or not. The path was meant to always start on the left and end on the right, not touching the top or bottom rows.

**Problem 13: Score the Test**

The trickiness of this problem lies in the fact that multiple key items can apply to a single student answer, but within each key item the student can only get credit once. So, setting up the right data structures to hold the key data is important. It’s tempting to try to manage things by problem number, but really each line of the key acts like its own entity so writing the code to check the answer just needs to take into account whether or not the student has already received points for the given key item.

**Problem 14: Shortest Flight Path**

This is the second exhaustive search problem. Any number of implementations would work here, but teams would have to search through all possible combinations to find the best one. You also have to watch out for circular paths to avoid an infinite loop situation.

**Problem 15: Pattern Part Play**

This problem looks insurmountable at first glance, but if a team happened to be very familiar with regular expressions it is definitely solvable. That was the point of having 15 patterns – we didn’t want to punish teams that didn’t know regular expressions, but teams that did would really need to work for their points. If you look at the solution provided, it’s really doing the same thing over and over just with different patterns. This was the first of two “real world” problems.

**Problem 16: Bizarro World Rover**

This problem looks easier than it is. The example inputs given in the problem packet illustrate one of the conditions requiring a team to fold the cube out in a non-obvious way. The math on this one is not killer, but thinking about how to rotate and unfold the cube is. We had two people solve this one, and both times it took far more than the two and a half hour limit. This was the first of our “try it at your own risk” problems.

**Problem 17: Pushing Blocks**

This was our second “high risk, high reward” problem. The general consensus from the team that generated the problem set was that this was the most difficult problem. As such, only one of our developers solved it. His solution is provided, along with a problem generator that he wrote to come up with boards to use for the judging inputs.

**Problem 18: Stuck-At Faults**

This is the second of the “real world” problems meant to show teams the types of problems developers are asked to solve in a corporate setting. This problem took the concept of logic operations and presented it in a way that hopefully the students had not seen before. The concepts are not difficult, but teams would need to consume and internalize the information quickly.