Example Lyft Laptop Interview

This is an example problem to get you familiar with the Lyft laptop interview process. On the day of your interview, you'll get a different, more complex problem.

Thanks for coming to interview with us today! You will be given 90 minutes to work on the below problem (15 minutes briefing, 60 minutes coding, 15 minutes debriefing). You will be coding independently, but your interviewer is available to answer questions during the whole process, either through chat or in person. You can also choose to have the interviewer in the room with you.

This question simulates solving a problem in a real world setting, separate from the whiteboard problems you'll solve in the rest of your interview. Since we're simulating the real world:

- You can use any language and environment you're comfortable with.
- You will submit your source code for your interviewer to review, compile, and grade.
- Consulting online resources and using open source libraries is **completely fair**. Use StackOverflow, Rosetta Code, and other sites to find resources. Reusing and adapting code fragments will not penalize your grade; we're grading your solution end-to-end as a whole, more than just the code that you wrote independently. Be sure to add a comment with the link of anything you adapted.

Also note that using a complete solution found online is **prohibited**, as this does not tell us anything about your coding ability.

Your goals should be to get a working and runnable solution, even if it's not complete. Your solution will be graded as follows:

1. **Correctness (40%)** - Your code should pass the sample test cases and other test cases you can think of. Handle edge cases sensibly. Don't worry about thread-safety or rogue input that doesn't adhere to the problem spec.
2. **Clean Code (35%)** - Your code should be split up into multiple classes or functions when relevant. It doesn't have to be in multiple files. It should have comments where relevant, but don't add comments just for the sake of commenting.
3. **Performance (25%)** - If you can make your solution's time and space complexity better, without sacrificing correctness, please do so!

Note: the above percent breakdown may be different for the question you get the day of your interview.

Remember, if you get really stuck, try and make progress however you can. Focus on simple cases, reduce the problem to more simpler components, and ask your interviewer for how to prioritize your time.

After the assignment, you will turn in your code to the interviewer. Your code will be graded by the interviewer and also by another blind interviewer who does not see your resume or anything about you.

Here's the problem:

Word Counter

Overview

Implement a program that counts occurrences of words in a given file.

The input will have multiple lines, with each line containing one or more words. The output will be each word and how many times it appears. You should print each word and its count on a separate line. The output should sort the words in lexicographically sorted order.

Sample Input

```
hello world
world hello
hello
howdy
```

Sample Output

```
hello 3
howdy 1
world 2
```

Notes & Tips

- If you have a laptop to code on, it is **strongly** recommended you bring one, so that you're familiar with the environment and toolchain. If not, we will provide you with a laptop running

Mac OS Sierra and pre-installed with vim, gcc, Python2, Ruby, Java, PyCharm, Sublime Text, & IntelliJ IDEA.

- Use the Lyft Guest WiFi network (the password will be provided the day of your interview).
- **For us to grade your solution**, your solution **MUST** read data from `stdin` and emit results to `stdout`. The emitted data must match the format described in the problem. Take a look at http://go.lyft.com/stdin for examples on reading from `stdin` and http://go.lyft.com/stdout for `stdout`.
- Compile early and often. Run your code against the sample input often to make sure your code continues to work as you make changes.
- Think about edge cases and other cases that aren't invoked in the sample input.
- Focus on correctness first before clean code and performance. Get your v1 working and passing our test cases, and any other cases you can think of, before iterating on refactoring code or making it more performant. We will value correctness above all when grading.