# Lyft Laptop Interview

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!

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:

# Versioned Key-Value Store

## Overview

You are to build a simple key-value store for storing integers (keys are strings, values are integers) and a **global** version (integer). You will not persist data to disk. You will store the data in memory.

The version number is an integer that increases monotonically. Every time **any key** is written with a value, the version number is increased. The first write is version number `1`. The second write is version number `2`, and so on.

The store supports three operations. The first is `PUT`, which returns the version number of this write. The second operation is the simple `GET`. This returns the **last** value mapped to the key. The third operation is the versioned `GET`. This takes a key and a version number and returns the value mapped to the key **at the time of the version number**. Assume that all inputs are *case sensitive*.

The input contains three types of commands corresponding to the three operations supported by the store:

1. `PUT <key> <value>`

   Set the key name to the value. Key strings will not contain spaces. Print out the version number, the key and the value as `PUT(#<version number>) <key> = <value>`. The first write in the file should be version number `1`, the second should be version number `2`, etc.

2. `GET <key>`

   Print out the key and the last value of the key, or `<NULL>` if that key has never been set as in: `GET <key> = <value>`

3. `GET <key> <version number>`

Print out the key, the version number and the value of key as it was at the time of the version number, as in `GET <key>(#version) = <value>`. If the key was not set at the time of the version number, print `<NULL>` for value. If the input version number is not recorded for a key, return the latest version recorded for that key that is smaller than the input version. See below for examples of formatted output.

## Sample Input

```
PUT key1 5
PUT key2 6
GET key1
GET key1 1
GET key2 2
PUT key1 7
GET key1 1
GET key1 2
GET key1 3
GET key4
GET key1 4
GET key2 1
```

## Sample Output

```
PUT(#1) key1 = 5
PUT(#2) key2 = 6
GET key1 = 5
GET key1(#1) = 5
GET key2(#2) = 6
PUT(#3) key1 = 7
GET key1(#1) = 5
GET key1(#2) = 5
GET key1(#3) = 7
GET key4 = <NULL>
GET key1(#4) = 7
GET key2(#1) = <NULL>
```

## Notes & Tips

- You can download sample files at https://go.lyft.com/interviews-versioned-kv-store (password: `enginterviews`).
- For WiFi, use the following:
  - NYC or WeWork: `WeWork` // `P@ssw0rd`
  - Otherwise: `Lyft Guest` // `iamaguest2day`
- **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. You may use code from http://go.lyft.com/stdin and http://go.lyft.com/stdout as a starting point.
- 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.
- Submit the zipped solution at http://go.lyft.com/submit-solution. You can do this in Finder by right-click > "Compress" or on the command line: `zip -r filename.zip foldername`. File name should be `<interview-question>-<date>.zip`. Use **interviewer's** name and email while uploading instead of candidate's.