Module 0

Exercise 0: Your own key-value store

1. Create a bash script named `mykv.sh`. This script accepts three parameters, `make`, `set` and `get`:
    1. `./mykv.sh make [table]` creates a `table` (actually an empty file in the current directory with the name supplied)
        - 💡 you could use `touch`
    2. `./mykv.sh set [table] [key] [value]` inserts a record with the supplied `key` and `value` and stores it on a line in the `table` file
        - 💡 you should use `>>`
    3. `./mykv.sh get [table] [key]` returns the current record with the supplied `key` in the `table`. If multiple records exist with the same key, it should only return the **last one**. If none exist, it should return `NULL`
        - 💡 you should use `grep`, `sed` and `tail`
2. Insert some records in a new table and check the functionality
    - What is the **worst-case** complexity of each operation?
    - Explain.
3. Implement a new endpoint: `./mykv.sh del [table] [key]` that deletes the record with the supplied `key` in the `table` so it will appear as `NULL` the next time you `get` it. Implement this as a `set` that sets the key to a special value that you will recognise as meaning *"this record is deleted"*.
    - 💡this is often referred to as a **tombstone**
    - What would have been another way to implement this?
4. Your database is now growing endlessly. To mitigate this, we will implement a `./mykv.sh cleanup [table]` that performs the actual deletion of data that is not needed anymore.
    - `cleanup` only keeps in the file the last version for every key
    - if the last version is a tombstone, it **drops it as well**
5. **Extension** Benchmark your implementation in terms of **average operations per second** for get, set and del
    - 💡you can use commands such as `openssl rand -hex 16` to generate random strings
6. **Advanced** Add a `sync -d` call after every write (using `&&`) to the file in the `set` code path to ensure **durability**
    - Read the `man sync`. Why does this help ensure durability?
    - Re-run your benchmarks. What do you see?
    - What do you think you should do to mitigate this?
7. **Advanced** You may now want to ensure that only one process can write to a table at the same time. Read about the `flock` syscall/utility and how you could use it guarantee this. Implement this in your routines. You may use a syntax that looks like the following for get, and a `-x` for `set`

```
(
    flock -s 42 && commands_executed_under_lock
) 42>/var/lock/mylock
```