Strawva 1

Your first programming project at FooCorp is to build an interactive shell that allows access to a transactional in-memory key / value store. For ease of use, this store will have a Read-Eval-Print loop (REPL) accessible via the command line that allows users to interactively write and query data in a session.

Implementation considerations

- Speed: Should optimize for read performance over writes
- Memory: Expect that this database will fit within the bounds of memory on a single machine.
- Durability: Data written to the database is not required to be durable beyond the interactive session.
- Multi-user: Multiple users are not required
- Code re-use: Please refrain from using off-the-shelf components for transactional storage (e.g. sqlite)

Trello, Basecamp, Asana, Kwip

Data Types

Keys and values within this system are simple alpha-numeric sequences with no spaces or special characters.

React.js 16
~70% Typescript

REPL Operations

In the following examples, ">" denotes the REPL prompt of the program, ***bolded*** *strings represent user input and normal values represent output from the program.*

Intellisense (completion)

`SET [KEY] [VALUE]`

Sets the given key to the specified value. If the key is already present, overwrite the old value.

```
> SET A 2
> GET A
A = 2
> SET A 3
> GET A
A = 3
```

eng team ~20
Zoom, Slack, email
✓

`DELETE [KEY]`

Deletes the given key. If the key has not been set, ignore.

```
> SET A 2
> GET A
A = 2
> DELETE A
> DELETE B
```

✓

```
> GET A
A not set
```

GET [KEY]
Prints out the current value of the specified key. If the key has not been set, prints a message.

```
> SET A 2
> GET A
A = 2
> GET B
B not set
```

COUNT [VALUE]
Returns the number of keys that have been set to the specified value. If no keys have been set to that value, prints 0.

```
> SET A FOO
> SET B BAR
> SET C FOO
> COUNT FOO
2
> COUNT BAR
1
> COUNT BAZ
0
```

BEGIN
Starts a transaction. These transactions allowing you to modify the state of the system and commit or rollback your changes. Transactions can be nested as well, as will be illustrated below.

COMMIT
Commits the changes made within the context of the active transaction and ends the active transaction. If no transaction is active, prints NO TRANSACTION

```
> COMMIT
NO TRANSACTION
> BEGIN
> SET A 1
> GET A
1
> COMMIT
```

```
> GET A
1
```

ROLLBACK

Throws away changes made within the context of the active transaction and ends the active transaction. If no transaction is active, prints NO TRANSACTION

```
> ROLLBACK
NO TRANSACTION
> BEGIN
> SET A 1
> GET A
1
> ROLLBACK
> GET A
A not set
```

Illustrative examples

Example 1:

```
> SET A 1
> SET B 1
> BEGIN
> SET A 2
> SET C 3
> COUNT 1
1
> ROLLBACK
> COUNT 1
2
> GET C
C not set
```

Example 2:

```
> SET A 1
> SET B 1
> BEGIN
> SET A 2
> DELETE B
> BEGIN
> COUNT 1
0
```

```
> DELETE A
> SET C 1
> COMMIT
> GET A
A not set
> COMMIT
> COUNT 1
1
```

Example 3:

```
> SET A 1
> SET B 1
> BEGIN
> SET A 2
> SET B 2
> COUNT 1
0
> BEGIN
> SET A 3
> COUNT 2
1
> ROLLBACK
> COUNT 2
2
> BEGIN
> DELETE B
> COUNT 2
1
> BEGIN
> GET B
B not set
> SET B 1
> COMMIT
> COUNT 2
1
> COMMIT
> GET B
1
> ROLLBACK
> GET A
1
> COUNT 1
2
> BEGIN
> BEGIN
> SET A 2
```

```
> SET A 3
> ROLLBACK
> GET A
1
> BEGIN
> SET A 4
> COMMIT
> GET A
4
> ROLLBACK
> GET A
1
```