

□ Simple Database Challenge

In the Simple Database problem, you'll implement an in-memory database similar to Redis. For simplicity's sake, instead of dealing with multiple clients and communicating over the network, your program will receive commands via standard input (stdin), and should write appropriate responses to standard output (stdout).

Guidelines

- This problem should take you between 30 and 90 minutes.
- We recommend that you use a high-level language, like Python, Go, Haskell, Ruby, or Java. We're much more interested in seeing clean code and good *algorithmic* performance than raw throughput.
- It is very helpful to the engineers who grade these challenges if you reduce external dependencies, make compiling your code as simple as possible, and include instructions for compiling and/or running your code directly from the command line, without the use of an IDE.
- Your submission must comply with the [input/output formats](#) and [performance requirements](#) specified below.

Data Commands

Your database should accept the following commands:

- `SET name value` – Set the variable `name` to the value `value`. Neither variable names nor values will contain spaces.
- `GET name` – Print out the value of the variable `name`, or `NULL` if that variable is not set.
- `UNSET name` – Unset the variable `name`, making it just like that variable was never set.
- `NUMEQUALTO value` – Print out the number of variables that are currently set to `value`. If no variables equal that value, print 0.
- `END` – Exit the program. Your program will always receive this as its last command.

Commands will be fed to your program one at a time, with each command on its own line. Any output that your program generates should end with a newline character. Here are some example command sequences:

INPUT	OUTPUT
SET ex 10	
GET ex	10
UNSET ex	
GET ex	NULL
END	

INPUT	OUTPUT
SET a 10	
SET b 10	
NUMEQUALTO 10	2

NUMEQUALTO 20	0
SET b 30	
NUMEQUALTO 10	1
END	

Transaction Commands

In addition to the above data commands, your program should also support database transactions by also implementing these commands:

- **BEGIN** – Open a new transaction block. Transaction blocks can be nested; a **BEGIN** can be issued inside of an existing block.
- **ROLLBACK** – Undo all of the commands issued in the *most recent* transaction block, and close the block. Print nothing if successful, or print **NO TRANSACTION** if no transaction is in progress.
- **COMMIT** – Close *all* open transaction blocks, permanently applying the changes made in them. Print nothing if successful, or print **NO TRANSACTION** if no transaction is in progress.

Any data command that is run outside of a transaction block should commit immediately. Here are some example command sequences:

INPUT	OUTPUT
BEGIN	
SET a 10	
GET a	10
BEGIN	
SET a 20	
GET a	20
ROLLBACK	
GET a	10
ROLLBACK	
GET a	NULL
END	

INPUT	OUTPUT
BEGIN	
SET a 30	
BEGIN	
SET a 40	
COMMIT	
GET a	40
ROLLBACK	NO TRANSACTION
END	

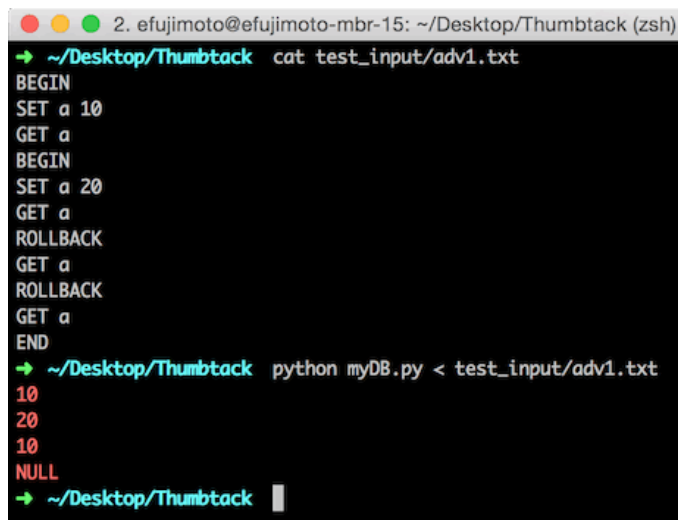
INPUT	OUTPUT
SET a 50	
BEGIN	
GET a	50
SET a 60	
BEGIN	
UNSET a	
GET a	NULL
ROLLBACK	
GET a	60
COMMIT	
GET a	60
END	

INPUT	OUTPUT
SET a 10	
BEGIN	
NUMEQUALTO 10	1
BEGIN	
UNSET a	
NUMEQUALTO 10	0
ROLLBACK	
NUMEQUALTO 10	1
COMMIT	

Input and Output Format

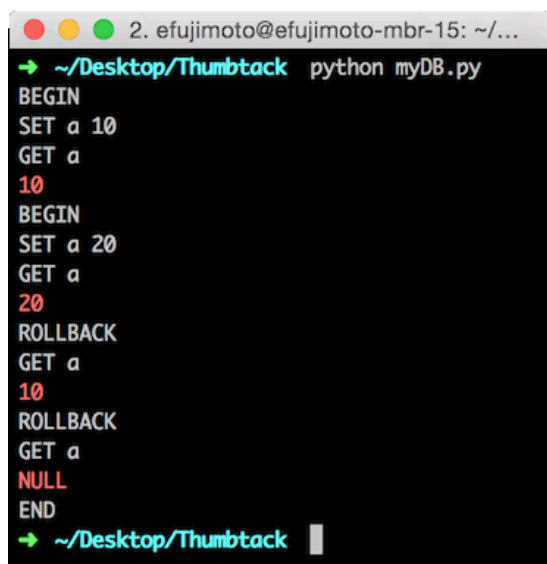
Your program should expect input from standard input (stdin) and handle EOF. Below are two examples of ways we might test your program.

- Pass a file of commands to standard input:



```
2. efujimoto@efujimoto-mbr-15: ~/Desktop/Thumbtack (zsh)
→ ~/Desktop/Thumbtack cat test_input/adv1.txt
BEGIN
SET a 10
GET a
BEGIN
SET a 20
GET a
ROLLBACK
GET a
ROLLBACK
GET a
END
→ ~/Desktop/Thumbtack python myDB.py < test_input/adv1.txt
10
20
10
NULL
→ ~/Desktop/Thumbtack
```

- Run the program interactively:



```
2. efujimoto@efujimoto-mbr-15: ~/...
→ ~/Desktop/Thumbtack python myDB.py
BEGIN
SET a 10
GET a
10
BEGIN
SET a 20
GET a
20
ROLLBACK
GET a
10
ROLLBACK
GET a
NULL
END
→ ~/Desktop/Thumbtack
```

The gray text is what we've typed in to the program and the red text is the program output. Please note that you do *not* need to implement any sort of text coloring. The different colors shown here are merely to help you distinguish between input and output.

Performance Requirements

- All of the following commands `BEGIN`, `GET`, `SET`, `UNSET`, and `NUMEQUALTO` should have an average-case runtime of $O(\log N)$ or better, where N is the total number of variables stored in the database.
- The vast majority of transactions will only update a small number of variables. Accordingly, your solution should be efficient about how much memory each transaction uses.

Insert mode