**Temporal in memory data store challenge**

You will be implementing a read-biased in-memory temporal data store.

A temporal store allows point-in-time queries about data. An example would be the price of an exchange traded security. We might capture the price of the security once an hour throughout the trading day, and store the price in a temporal store. We can then query the temporal store to determine at any given time, what the most recent captured price was. This example illustrates the following key terms, which will be used throughout this document:

1. Identifier - a key (e.g. the security id), for which we want to store a history of. observations
2. History - the list of observations for a given identifier
3. Observation - a piece of data which is associated with a particular identifier at a particular timestamp
4. Timestamp - a given point in time

The latest observation for an identifier at a timestamp is found by searching in the identifier's history for the first observation whose timestamp is less than, or equal to, the sought timestamp.

For the purposes of this scenario, you should assume the following:

1. identifiers are integers in the inclusive range [0..2^31 - 1]
2. timestamps are integers in the inclusive range [0..2^63 - 1]
3. data is represented as a contiguous string of non-whitespace characters. For example, "jiggawatt" is a valid piece of data. The string "great scot" is not, owing to its whitespace.
4. users may interact with the temporal store, as well as processes. you should be strict in what data you accept, but should provide suitable error messages where the user has erred.
5. capacity; there is no requirement to restrict your store to a given size, but you may assume that you will be provided with no more than 10,000 identifiers, each with no more than 1,000 history entries, and no data item will be no more than 16 characters.

Your temporal data store will communicate with other processes via standard input and output. A series of commands are read and applied to the data store from standard input. The results of evaluating the commands should be written to standard output. A command will only ever be one line; the result of executing a command will be a single output line, except for the QUIT command. When there are no more lines to consume, your temporal data store should exit. There is no need to persist data in the store between executions (this is an in-memory temporal store).

The commands which can be executed against your data store are described below. Note that:

* items in <angle brackets> represent required arguments, but the angle brackets themselves do not form part of the command
* items in [square brackets] are optional arguments successfully processed commands should always start their response with "OK ", followed by the result of executing the command
* commands which could not be executed should always start their response with "ERR ", followed by a reasonable error message.
* commands are not sensitive to whitespace; arguments must be separated by at least one character of whitespace, but may be separated by more.

**CREATE <id> <timestamp> <data>**

* Creates a new history for the given identifier, if there is no existing history. Adds an observation to the newly created history for the given timestamp and data. CREATE should not be executed if the provided identifier already has a history. Returns the data which was inserted, to confirm insertion.

**UPDATE <id> <timestamp> <data>**

* Inserts an observation for the given identifier, timestamp and data. Returns the data from the prior observation for that timestamp.

**DELETE <id> [timestamp]**

* If timestamp is provided, deletes all observations for the given identifier from that timestamp forward. Returns the current observation at the given timestamp, or an error if there is no available observation.
* If timestamp is not provided, deletes the history for the given identifier, and returns the observation with the greatest timestamp from the history which has been deleted.

**GET <id> <timestamp>**

* Returns the data from the observation for the given identifier at the given timestamp, or an error if there is no available observation.

**LATEST <id>**

* Locates the observation with the greatest timestamp from the history for the given identifier, and and data.

**QUIT**

* Terminates the process immediately. No response should be written.

An example interaction with your temporal data store might look like this:

**CREATE 0 100 1.5**

**OK 1.5**

**UPDATE 0 105 1.6**

**OK 1.5**

**GET 0 100**

**OK 1.5**

**GET 0 110**

**OK 1.6**

**LATEST 0**

**OK 105 1.6**

**LATEST 1**

**ERR No history exists for identifier '1'**

**CREATE 1 110 2.5**

**OK 2.5**

**CREATE 1 115 2.4**

**ERR A history already exists for identifier '1'**

**UPDATE 1 115 2.4**

**OK 2.5**

**UPDATE 1 120 2.3**

**OK 2.4**

**UPDATE 1 125 2.2**

**OK 2.3**

**LATEST 1**

**OK 125 2.2**

**GET 1 120**

**OK 2.3**

**UPDATE 1 120 2.35**

**OK 2.3**

**GET 1 122**

**OK 2.35**

**DELETE 1 122**

**OK 2.35**

**GET 1 125**

**OK 2.35**

**DELETE 1**

**OK 2.35**

**QUIT**