Backend Exercise

The following exercise should be completed in no more than 4 hours. You should submit it as a zip / tarball via email. If you have any questions at any point during the exercise, please reach out to zach@frame.io.

Background

We'd like to make a **fact engine**. Ultimately, of course, this should be a computer program that will contain all the world's knowledge; but we can start small.

The MVP of our program will have two functions: the ability to **input** facts and the ability to **query** them.

Here's an example of a fact: *Lucy is a cat.* We can represent that like this: `is_a_cat (lucy)`.

Here's another example of a fact: *Alex and Sam are friends.* We can represent that like this: `are_friends (alex, sam)`. In general, we can represent a fact as a **statement** (the first element of the fact) about *one or more* **arguments** (the elements inside of the parentheses).

The first thing we can do with our program is input those facts. It should record everything we input for later reference. We could represent a session of inputting some facts like this:

```
INPUT is_a_cat (lucy)
INPUT is_a_cat (garfield)
INPUT are_friends (alex, sam)
INPUT is_a_cat (bowler_cat)
INPUT are_friends (frog, toad)
```

The second thing we can do is make queries. For instance, we can ask it *is Lucy a cat?*. We can represent that like this:

```
QUERY is_a_cat (lucy)
```

We would expect this response:

```
---
true
```

(responses have `---` at the top). Lucy is a cat.

On the other hand if we asked for a fact that our program didn't know about:

```
QUERY is_a_cat (alf)
```

We'd expect the answer to be

```
---
false
```

Insofar as we know, Alf is not a cat.

In order to be truly useful, our fact engine needs to be able to answer questions using the facts that it has. We want to be able to ask *who is friends with Sam?* or *who is a cat?*. So the last part of our query functionality is that we can put placeholders in our queries. We'll use capital letters to stand for a variable we're asking about. For instance, query *who is friends with Sam?* we would enter:

```
QUERY are_friends (X, sam)
```

In this case the response would be

```
---
X: alex
```

There was one fact that matched that query, and the program returns the part of the fact that matches up with `X`.

If there are multiple facts that match, it will return all of them. Asking *who is a cat?*:

```
QUERY is_a_cat (X)
```

We expect to get back all the facts that match the query:

```
---
X: lucy
X: garfield
X: bowler_cat
```

We can also ask *what are the pairs of friends we know about?*:

```
QUERY are_friends (X, Y)
```

And get:

```
---
X: alex, Y: sam
X: frog, Y: toad
```

Finally, it will be useful to make placeholders stand for the same thing if they appear multiple times. For instance, we want to communicate that Sam is friends with themselves (as indeed we all should be).

```
INPUT are_friends (sam, sam)
```

Then if want to ask *who is friends with themselves?*, with

```
QUERY are_friends (Y, Y)
```

We expect to get back:

```
---
Y: sam
```

Assignment

Your assignment is to write a program that can take as input a file containing `INPUT` or `QUERY` commands, one per line, and process them in order.

When the program hits an `INPUT` line it should update its internal state. When it hits a `QUERY` line it should print the output to standard out. Since query output can be multiple lines, each output should be preceded by three hyphens: `---`.

We've provided an `examples/` directory with a few test cases that you can use to write your program. Your program should be able to ingest the `in.txt` files and write to standard out what is contained in `out.txt`. We'll use these test cases to evaluate your program, as well as a separate set of test cases that only we have access to.

Please make it simple and straightforward for a reviewer to set up and run your project on their computer. This includes including setup instructions in a README file. You may use any programming language you like, though if your language requires the presence of an interpreter or compiler on the reviewer's machine we ask that you restrict yourself to languages that can be acquired through standard package managers like Homebrew or Arch/AUR.

Criteria

We'll be evaluating your program according to several criteria:

- Was a reviewer able to easily and quickly run your code on their computer?
- Does it provide the expected responses to well-formed input?
- How does it handle errors and invalid input?
- How readable is the code?
- How efficient is the code?
- How well-structured is the application source?
- How well-documented is it?
- How well-tested is it?

We'll also be interested in discussing with you, either over the phone or in person, some of the tradeoffs and design decisions you made when writing your program.