COMP 348: Principles of Programming Languages

Assignment 3

Summer 2023, sections AA and AB

June 16, 2023

Functional Programming with Clojure

Date posted: Friday, June 16th, 2023.

Date due: Wednesday, June 28th, 2023, by 23:59.

Weight: $8\frac{1}{3}\%$ of the overall grade.

Overview

In this assignment you implement a very simple query system using Clojure. REALLY simple. In fact, all it will really do is load data from a series of text files. This data forms your "in-memory database".

Database Schema

Each table will have a "schema" that indicates the fields inside.

Students

This is the data file for the students table. The schema is:

```
<studID, name, address, phoneNumber>
```

An example of the studs.txt disk file might be:

```
2408011|John Smith|123 Here Street|514-456-4567
4208071|Sue Jones|43 Rose Court Street|514-345-7867
2038990|Fan Yuhong|165 Happy Lane|514-345-4533
```

Courses

This is the data file for the courses table. The schema is:

```
   <id, cournam, courno, credits, desc>
```

An example of the courses.txt disk file might be:

```
1|COMP|348|3|Principles of Programming Languages
2|COMP|335|3|Theoretical Computer Science
3|COMP|232|3|Discrete Math.
4|COMP|111|1|Some made-up course!
```

Grades

This is the data file for the grades table. The schema is:

```
   <studID, courseID, semester, grade>
```

An example of the grades.txt disk file might be:

```
2408011|1|2023-S|A+
2408011|2|2023-S|A
2408011|3|2023-S|B+
4208071|1|2023-S|A-
4208071|2|2023-S|B+
2038990|3|2023-S|B
2038990|4|2023-S|A+
```

System Menu

You will provide the following menu to allow the user to perform actions on the data:

```
*** SIS Menu ***
------------------
1. Display Courses
2. Display Students
3. Display Grades
4. Display Student Record
5. Calculate GPA
6. Course Average
7. Exit
Enter an option?
```

The first 3 options simply display the content of the in-memory tables. A sample output of displaying students should be similar (not necessarily identical) to the following:

```
["2408011" "John Smith" "123 Here Street" "514-456-4567"]
["4208071" "Sue Jones" "43 Rose Court Street" "514-345-7867"]
["2038990" "Fan Yuhong" "165 Happy Lane" "514-345-4533"]
```

The Student Record should list all courses that the student has taken. Upon asking for the student id the system displays student id and name followed by the courses.
The ID values aren't very useful for viewing purposes, so the course names and descriptions must be displayed instead of the internal ids. A sample output is give in the following:

```
["2408011" "John Smith"]
["COMP 348" "Principles of Programming Languages" "2022-S" "A+"]
["COMP 335" "Theoretical Computer Science" "2022-S" "A"]
["COMP 232" "Discrete Math." "2022-S" "B+"]
```

The courses are ordered by course names and numbers

For option 5, the GPA for the student is calculated by applying a weighted average on the course grades, where the number of credits would be used for the weights. The grade letters may be converted into numerical values using the following map:

```
{
  "A+" 4.3, "A" 4, "A-" 3.7,
  "B+" 3.3, "B" 3, "B-" 2.7,
  "C+" 2.3, "C" 2, "C-" 1.7,
  "D+" 1.3, "D" 1, "D-" 0.7,
  "F"  0 }
```

The Course Average may be simply calculated by converting all letter grades to numerical values (using the above map) and computing the average per semester.

```
["COMP 232" "2023-S" "3.15"]
```

Operations

Here are some additional points to keep in mind:

- You do not want to load the data each time a request is made. So, before the menu is displayed the first time, your data should be loaded and stored in appropriate data structures. So, assuming a `loadData` function in a module called `db`, an invocation like the following would create and load a data structure called `studDB`:

  ```
  (def studDB (db/loadData "studs.txt"))
  ```

  The studDB data structure can then be passed as input to any function that needs to act on the data. Note that, once it is loaded, the data is never updated.

- As a functional language, Clojure uses recursion. If possible, use the `map`, `reduce` and `filter` functions whenever you can (Note that you may sometimes have to write your own recursive functions when something unique is required).

- This is a Clojure assignment, not a Java assignment. So Java should not be used for any main functionality. That said, it might be necessary to use Java classes to convert text to numbers in order to do the sales calculations. An example with the map function is shown below:

  ```
  (map #(Integer/parseInt %) ["6" "2" "3"]))
  ```

- It is worth noting, however, that this can also be done with Clojure's `read-string` function. This can be used to translate "numeric" strings, including floating point values. So we could do something like:

  ```
  (* 4 (read-string "4.5")) ; = 18
  ```

- The I/O in this assignment is trivial. While it is possible to use complex I/O techniques, it is not necessary to read the text files. Instead, you should just use slurp, a Clojure function that will read a text file into a string, as in:

  ```
  (slurp "myFile.txt")
  ```

- For the input from the user, the `read-line` function can be used. If you print a prompt string to the screen (e.g., "please enter a name", you may want to use `flush` before `read-line` so that the prompt is not cached (and therefore not displayed).

- For string processing, you will want to use `clojure.string`. You can view the online docs for a list of string functions and examples (1).

- Do not worry about efficiency. There are ways to make this program (both the data management and the menu) more efficient, but that is not our focus here. We just want you to use basic functionality to try to get everything working.

Project Environment

It is easy to run a single Clojure file from the command line (i.e., `clojure myfile.clj`). It becomes a little more tedious when the app is made up of multiple files. In practice, large projects are typically built with a tool called Leinengen. Use of Leiningen, however, is overkill for this kind of assignment and will likely make building and testing more problematic for most students. We will therefore keep things as simple as possible. Your files, both source code and data will be located in the current folder. Each of the three source files listed above will define its own namespace. For example:

```
(ns app
  (:require [db])
  (:require [menu]))
```

IMPORTANT: This will allow the files/modules to interact with one another. However, Clojure accesses modules relative to the current `CLASSPATH`. By default, the `CLASSPATH` will probably not be set on your machine, so Clojure will fail to run your program, with errors saying that it cannot find your modules. The simplest thing to do is to simply set the `CLASSPATH` to include the current folder (this is what the graders will do). From the Linux command line, you can just say

```
export CLASSPATH=./
```

Once this is done, you can run your app simply by typing

```
clojure app.clj
```

Note that this `CLASSPATH` variable is temporary and would have to be reset each time you start Linux. If you want it to be automatically configured every time you log in, you could add the export line to the `.bashrc` file in your login folder.

Submission

Your submission will have just 3 source files. The “main” file will be called `app.clj` and will be used to simply load the information in the text files into a group of data structures and then pass this data to the function that will provide the initial functionality for the app. The second file, `menu.clj` will, as the name implies, provide the interface to the user. The third file will be called `db.clj` and will contain all of the data management code (loading, organizing, etc.). Do not include any data files with your submission, as the markers will provide their own.

Once you are ready to submit, place the .clj files into a zip file. The name of the zip file will consist of “a3” + last name + first name + student ID + ".zip", using the underscore character “_” as the separator. For example, if your name is John Smith and your ID is “123456”, then your zip file would be combined into a file called “`a2_Smith_John_123456.zip`”. The final zip file will be submitted through the course web site on Moodle. You simply upload the file using the link on the assignment web page.

Please note that it is your responsibility to check that the proper files have been uploaded. No additional or updated files will be accepted after the deadlines. You can not say that you accidentally submitted an “early version” to Moodle. You are graded only on what you upload.

Note that assignments must be submitted on time in order to receive full value. Appropriate late penalties ($<$ 1 hour = 10%, $<$ 24 hours = 25%) will be applied, as appropriate.

References

1. Clojure String:

   `https://clojuredocs.org/clojure.string`