CMPE 260 - Spring 2018
Scheme Programming Assignment
Due: 07.05.2018 23:55

1 Introduction

This project will test your knowledge of functional programming and Scheme. Your code will be graded on correctness, readability, testing strategy, efficiency, documentation and the use of functional programming style. Your task is to write Scheme functions (statements) for manipulating lists of travelers and locations.

2 Problem Description

You will extract different sorts of information from a database of locations and travelers. In Scheme, the database is encoded as a list of entries, with one entry for each location and again one entry for each traveler. Each entry will be a list in the following form:

(<location> <accommodation-cost> (<neighbor-location1> ... <neighbor-locationN>) (<activity1> ... <activityN>))

which indicates that <location> is a city which has a railway connection with cities <neighbor-location1> ... <neighbor-locationN> and hosts activities <activity1> ... <activityN> and staying one night in <location> costs <accommodation-cost>.

(<traveler> (<city1> ... <cityN>) (<activity1> ... <activityN>) <home>)

which indicates that <traveler> is a traveler who likes to visit cities <city1> . . . <cityN> and enjoys activities <activity1> ... <activityN> and who lives in <home>.

A small sample database is as follows:

```
(define LOCATIONS
    '(
        (newyork 100 (ohio indiana newjersey) (theatre concert opera))
        (california 120 (washington utah) (theatre))
        (ohio 75 (newyork indiana newjersey) (concert))
        (moscow 95 () (concert opera))
        (paris 150 (nice cannes) (concert opera))
        (copenhagen 95 (nuenen) (theatre concert opera))
        (texas 80 (utah illinois indiana) (theatre concert))
        (cambridge 90 (cork london nuenen brussels) (theatre))
        (brussels 90 (london cambridge nuenen paris vienna) (theatre concert opera))
        (newjersey 100 (ohio newyork) (theatre concert))))

(define TRAVELERS
    '(
        (john (ohio texas) (theatre concert opera) newyork)
        (james (texas ohio copenhagen) (theatre concert opera) newjersey)
        (richard (cambridge ohio texas) (theatre concert) california)
        (alan (california ohio) () cambridge)
        (mary (california) (concert) cambridge)
        (ingrid (moscow ohio texas) (opera) brussels)))
```

Remind that:

- All parts are required in all entries.
- If the location has no railway network, then (<neighbor-location1> ... < neighbor-locationN>) is an empty list.
- If the location has no activities, then (<activity1> ... <activityN>) is an empty list.
- If the traveler has no interested cities, then (<city1> ... <cityN>) is an empty list.
- If the traveler has no interested activities, then (<activity1> ... <activityN>) is an empty list.

After loading this define statements, you can treat LOCATIONS and TRAVELERS as global variables whose values are the database lists.

3 Operations

3.1 Simple Queries

Write the following functions:

- **RAILWAY-CONNECTION**: takes a single argument (a location) and returns the list involving the neighbor locations.

```
> (RAILWAY-CONNECTION 'newyork)
(list 'ohio 'indiana 'newjersey)
> (RAILWAY-CONNECTION 'brussels)
(list 'london 'cambridge 'nuenen 'paris 'vienna)
> (RAILWAY-CONNECTION 'moscow)
'()
> (RAILWAY-CONNECTION 'city_not_in_database)
'()
```

- **ACCOMMODATION-COST**: takes a single argument (a location) and returns one night accommodation cost in that location.

```
> (ACCOMMODATION-COST 'texas)
80
> (ACCOMMODATION-COST 'city_not_in_database)
0
```

- **INTERESTED-CITIES**: takes a single argument (a traveler) and returns a list involving his/her interested cities.

```
> (INTERESTED-CITIES 'john)
(list 'ohio 'texas)
> (INTERESTED-CITIES 'richard)
(list 'cambridge 'ohio 'texas)
```

- **INTERESTED-ACTIVITIES**: takes a single argument (a traveler) and returns a list involving his/her interested activities.

```
> (INTERESTED-ACTIVITIES 'john)
(list 'theatre 'concert 'opera)
> (INTERESTED-ACTIVITIES 'richard)
(list 'theatre 'concert)
> (INTERESTED-ACTIVITIES 'alan)
'()
```

- **HOME**: takes a single argument (a traveler) and returns his/her hometown.

```
> (HOME 'john)
'newyork
> (HOME 'richard)
'california
```

3.2 Constructing Lists

Write the following functions:

- **TRAVELER-FROM**: takes a single argument (a location) and returns a list involving the travelers whose hometown is that location.

```
> (TRAVELER-FROM 'newyork)
(list 'john)
> (TRAVELER-FROM 'cambridge)
(list 'alan 'mary)
> (TRAVELER-FROM 'city_not_in_database)
'()
```

- **INTERESTED-IN-CITY**: takes a single argument (a city) and returns a list of travelers who wants to visit that city.

```
> (INTERESTED-IN-CITY 'california)
(list 'alan 'mary)
> (INTERESTED-IN-CITY 'cambridge)
(list 'richard)
> (INTERESTED-IN-CITY 'city_not_in_database)
'()
```

- **INTERESTED-IN-ACTIVITY**: takes a single argument (an activity) and returns a list of travelers who enjoy that activity.

```
> (INTERESTED-IN-ACTIVITY 'concert)
(list 'john 'james 'mary)
> (INTERESTED-IN-ACTIVITY 'opera)
(list 'john 'james 'ingrid)
> (INTERESTED-IN-ACTIVITY 'activity_not_in_database)
'()
```

3.3 Connected Cities

Write the RAILWAY-NETWORK function which takes a single argument (a location) and returns a list of locations which are accessible from this location by train. The list excludes the given location.

```
> (RAILWAY-NETWORK 'newyork)
(list 'ohio 'indiana 'newjersey)
> (RAILWAY-NETWORK 'brussels)
(list 'london 'cambridge 'nuenen 'paris 'vienna 'cork 'nice 'cannes)
> (RAILWAY-NETWORK 'moscow)
'()
> (RAILWAY-NETWORK 'city_not_in_database)
'()
```

3.4 Expenses

In holidays, a traveler visits one of his/her interested cities. But this trip requires funding. Write the following functions:

- **ACCOMMODATION-EXPENSES**: takes two arguments (a traveler and a location) and returns the accommodation cost for the traveler. If the location is his/her hometown, then no funding is necessary. If the location is another city, then the traveler stays in hotel where one night cost is given in the LOCATIONS database. If the location hosts an interesting activity for the traveler, then the traveler stays for three nights to enjoy more, otherwise he/she stays only for one night.

```
> (ACCOMMODATION-EXPENSES 'john 'newyork)
0
> (ACCOMMODATION-EXPENSES 'john 'ohio)
225
> (ACCOMMODATION-EXPENSES 'john 'texas)
225
> (ACCOMMODATION-EXPENSES 'mary 'california)
120
```

- **TRAVEL-EXPENSES**: takes two arguments (a traveler and a location) and returns the travel cost for the visitor. If the visited city resides in the railway network of the traveler's hometown, then it costs only 50 units ("Aktarma" between trains is free of charge). Otherwise, the traveler uses airlines which cost 100 units. Recall that the traveler uses the same way to return back to home.

```
> (TRAVEL-EXPENSES 'john 'newyork)
0
> (TRAVEL-EXPENSES 'john 'ohio)
100
> (TRAVEL-EXPENSES 'john 'texas)
200
> (TRAVEL-EXPENSES 'mary 'california)
200
```

- **EXPENSES**: takes two arguments (a traveler and a location) and returns the sum of travel cost and accommodation cost for the traveler.

```
> (TRAVEL-EXPENSES 'john 'newyork)
0
> (TRAVEL-EXPENSES 'john 'ohio)
325
> (TRAVEL-EXPENSES 'john 'texas)
425
> (TRAVEL-EXPENSES 'mary 'california)
320
```

3.5 Categorizing Cities

Write the IN-BETWEEN function which takes two arguments (two limits) and returns the list of cities whose accommodation costs are between these limits, inclusively.

```
> (IN-BETWEEN 110 120)
(list 'california)
> (IN-BETWEEN 80 90)
(list 'texas 'cambridge 'brussel)
> (IN-BETWEEN 300 400)
'()
```

4 Submission Details

You should use DrRacket for implementation. The first two lines of your program should be

```
#lang scheme
; student-id
```

You should submit only one file on Moodle, which should be named as

```
traveler_solution.rkt
```

Your Scheme program should operate on a database as illustrated above. A complete sample database called travelerdb.rkt will be provided; you can use this database for testing your program. Put all definitions of the functions above, as well as any other functions, in a file called traveler_solution.rkt. Do NOT include the define statement for the LOCATIONS and TRAVELERS databases in the file you submit. Submit only your traveler_solution.rkt file on Moodle.

Your assignment will be marked not only for correctness, but for functional programming style, comments, and your testing strategy as well. Furthermore, we will test the code that you submit electronically on our own test cases, using new LOCATIONS and TRAVELERS databases. For that reason, you must use the exact function names and argument lists that we have specified, and use the exact file name and submit instructions given above.

5 Important Notes

- The project will be done individually, not as a group.
- The following language constructs are explicitly prohibited. You will not get any points if you use them:
  - Any function or language element that ends with an !.
  - Any of these constructs: begin, begin0, when, unless, for, for*, do, loop, set!-values.
  - Any language construct that starts with for/ or for*/.
  - Any construct that causes any form of mutation (impurity).
- In short, you must follow pure functional programming style.
- You can use Racket reference, either from DrRacket's menu: Help > Racket Documentation, or from the following link: http://docs.racket-lang.org/reference/index.html
- Simply Scheme: Introducing Computer Science is a nice book for learning Scheme: https://people.eecs.berkeley.edu/ bh/ss-toc2.html
- SICP (Structure and Interpretation of Computer Programs a.k.a. The Purple Book) is a nice book for learning Scheme: https://mitpress.mit.edu/sicp/full-text/book/book.html
- There are also video lectures using purple book on MIT OCW, by the authors: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-001-structure-and-interpretation-of-computer-programs-spring- video-lectures/