CMPE 260 - Spring 2019
Scheme Programming Project
Due: 17.05.2018 23:55

1 Introduction

This project will test your knowledge of functional programming and Scheme. Your code will be graded on correctness and the use of functional programming style. Your task is to write Scheme functions (statements) for manipulating lists of farms, customers and crops.

2 Problem Description

You will extract different sorts of information from a database of farms, customers and crops in an agricultural area. In Scheme, the database is encoded as a list of entries, with one entry for each farm, one entry for each customer and multiple entries for crops. Each entry will be a list in the following form:

```
(<farm> <transportation-cost> (<crop1> ... <cropN>))
```

which indicates that <farm> is a farm which grows the crops <crop1> ... <cropN> and transportation of the bought crop to the customer costs <transportation-cost>.

```
(<customer> (<farm1> ... <farmN>) (<crop1> ... <cropN>))
```

which indicates that <customer> is a customer who has contract with farms <farm1> . . . <farmN> and wants to buy crops <crop1> ... <cropN>.

```
(<crop> <farmX> <sale-price>)
```

which indicates that <crop> is a crop which is sold on <sale-price> at <farmX>.

A small sample database is as follows:

```
(define FARMS
    '(
        (farmA 100 (apricot apple blueberry))
        (farmB 90 (broccoli carrot grape))
        (farmC 75 (corn grape lemon))
        (farmD 75 ())))

(define CUSTOMERS
    '(
        (john (farmA farmC) (apricot lemon))
        (james (farmB farmC) (grape corn))
        (arya (farmB farmD) (grape broccoli))
        (elenor () ())))

(define CROPS
    '(
        (apricot farmA 10)
        (apple farmA 12)
        (blueberry farmA 15)
        (broccoli farmB 8)
        (carrot farmB 5)
        (grape farmB 10)
        (corn farmC 9)
        (grape farmC 12)
        (lemon farmC 10)))
```

Remind that:

- All parts are required in all entries.
- If the farm has no crops, then (<crop1> ... <cropN>) is an empty list.
- If the customer has no contract with any farms, then (<farm1> ... <farmN>) is an empty list.
- If the customer has no crops to buy, then (<crop1> ... <cropN>) is an empty list.

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

3 Operations

3.1 Simple Queries

Write the following functions:

- **TRANSPORTATION-COST**: takes a single argument (a farm) and returns the transportation cost of crops from that farm.

```
> (TRANSPORTATION-COST 'farmA)
100
> (TRANSPORTATION-COST 'farm_not_in_database)
0
```

- **AVAILABLE-CROPS**: takes a single argument (a farm) and returns a list of available crops that are grown on that farm.

```
> (AVAILABLE-CROPS 'farmA)
(apricot apple blueberry)
> (AVAILABLE-CROPS 'farm_not_in_database)
()
```

- **INTERESTED-CROPS**: takes a single argument (a customer) and returns a list involving his/her interested crops.

```
> (INTERESTED-CROPS 'john)
(apricot lemon)
> (INTERESTED-CROPS 'elenor)
()
> (INTERESTED-CROPS 'customer_not_in_database)
()
```

- **CONTRACT-FARMS**: takes a single argument (a customer) and returns a list farms that the customer has contract with.

```
> (CONTRACT-FARMS 'john)
(farmA farmC)
> (CONTRACT-FARMS 'elenor)
()
> (CONTRACT-FARMS 'customer_not_in_database)
()
```

3.2 Constructing Lists

Write the following functions:

- **CONTRACT-WITH-FARM**: takes a single argument (a farm) and returns a list of customers who has contract with that farm.

```
> (CONTRACT-WITH-FARM 'farmA)
(john)
> (CONTRACT-WITH-FARM 'farmB)
(james arya)
> (CONTRACT-WITH-FARM 'farm_not_in_database)
()
```

- **INTERESTED-IN-CROP**: takes a single argument (a crop) and returns a list of customers who wants to buy that crop.

```
> (INTERESTED-IN-CROP 'apricot)
(john)
> (INTERESTED-IN-CROP 'grape)
(james arya)
> (INTERESTED-IN-CROP 'crop_not_in_database)
()
```

3.3 Shopping Prices

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

- **MIN-SALE-PRICE**: takes a single argument (a crop) and returns the minimum sale price for that crop.

```
> (MIN-SALE-PRICE 'apricot)
10
> (MIN-SALE-PRICE 'grape)
10
> (MIN-SALE-PRICES 'crop_not_in_database)
0
```

- **CROPS-BETWEEN**: takes two arguments (a min price and a max price) and returns the list of crops which has sale price between these arguments, inclusive.

```
> (CROPS-BETWEEN 11 15)
(apple blueberry grape)
> (CROPS-BETWEEN 100 200)
()
```

- **BUY-PRICE**: takes two arguments (a customer and a crop) and returns the minimum cost of buying that crop for the customer. The customer can buy the crop from a farm which has a contract with the customer and also grows the crop. The buy price is the sum of the sale price for that crop and the transportation cost from the farm.

```
> (BUY-PRICE 'john 'appricot)
110
> (BUY-PRICE 'james 'corn)
84
> (BUY-PRICE 'james 'grape)
87
```

- **TOTAL-PRICE**: takes a single argument (a customer) and returns the minimum total price for buying all the crops in the customers' list. The transportation cost is calculated separately for each crop. This means that, even if the customer buys two crops from the same farm, the customer will pay separate transportation costs for each crop, a total of two times the transportation cost of the farm.

```
> (TOTAL-PRICE 'john)
195
> (TOTAL-PRICE 'james)
171
> (TOTAL-PRICE 'arya)
198
> (TOTAL-PRICE 'elenor)
0
```

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

```
grocery_solution.rkt
```

Your Scheme program should operate on a database as illustrated above. A complete sample database called grocerydb.rkt will be provided; you can use this database for testing your program. Do NOT include the define statements for the FARMS, CUSTOMERS and CROPS databases in the file you submit. Use the exact function names and argument lists that we have specified, and use the exact file name and submit instructions given above. Do not use same function names for your your helping functions.

5 Important Notes

- The project will be done individually, not as a group.
- There will be a question thread in Piazza for this project. Questions sent in any other way or thread will not be taken into account. Questions will be answered in working hours.
- 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/