CmpE 260 - Principles of Programming Languages
Summer 2019
Scheme Programming Project Description

due: 07.08.2019 - 08:59

1 Introduction

A bunch of hotels order baked goods from bakeries that they have an agreement with. Your task is to write the Scheme code to answer certain queries given a database of hotels, bakeries, and baked goods.

2 Problem Description

You will extract different sorts of information from a database of bakeries, hotels, and baked goods. In Scheme, the database is encoded as a list of entries, with one entry for each bakery, one entry for each hotel and multiple entries for baked goods. Each entry will be a list in the following form:

(<good> <bakery> <price>)

which indicates that `<good>` is a baked good which is sold for `<price>` at `<bakery>`.

(<bakery> <delivery-cost> (<good1> ... <goodN>))

which indicates that `<bakery>` is a bakery which sells the baked goods `<good1> ... <goodN>` and the delivery of the goods to the hotel costs `<delivery-cost>`.

(<hotel> (<bakery1> ... <bakeryN>) (<good1> ... <goodN>))

which indicates that `<hotel>` is a hotel who has an agreement with bakeries `<bakery1>... <bakeryN>` and wants to buy baked goods `<good1> ... <goodN>`.

```
(define GOODS
      (
      (borek hasanpasa 12)
      (corek hasanpasa 4)
      (kurabiye hasanpasa 15)
      (pogaca aynali 3)
      (kek aynali 5)
      (simit aynali 2)
      (kepekli istinye 6)
      (simit istinye 3)
      (portakalli istinye 3)))

(define BAKERIES
```

```
    (
    (hasanpasa 50 (borek corek kurabiye))
    (aynali 60 (pogaca kek simit))
    (istinye 70 (kepekli simit portakalli))
    (sariyer 80 ()))

(define HOTELS
    (
    (hilton (hasanpasa istinye) (borek portakalli))
    (conrad (aynali istinye) (simit kepekli))
    (hyatt (aynali sariyer) (simit pogaca))
    (marmara () ())))
```

- All parts are required in all entries.
- If the bakery has no goods, then `(<good1> ... <goodN>)` is an empty list.
- If the hotel has no contract with any bakeries, then `(<bakery1> ... <bakeryN>)` is an empty list.
- If the hotel has no goods to buy, then `(<good1> ... <goodN>)` is an empty list.

After loading this define statements, you can treat BAKERIES, HOTELS and GOODS as lists whose values are the database lists.

3 Operations

3.1 Basic Queries

Write the following functions (use exactly the same name):

1. DELIVERY-COST: takes a single argument (a bakery) and returns the delivery cost of that bakery (e.g. 50 for `(DELIVERY-COST 'hasanpasa)`). Returns 0 for a bakery that does not exist in the database.
2. GOODS-AVAILABLE: takes a single argument (a bakery) and returns a list of baked goods that are sold by that bakery (e.g. `(borek corek kurabiye)`) for `(GOODS-AVAILABLE 'hasanpasa)`). Returns empty list for a bakery that does not exist in the database.
3. GOODS-INTERESTED: takes a single argument (a hotel) and returns a list of goods that it would like to buy (e.g. `(borek portakalli)`) for `(GOODS-INTERESTED 'hilton)`). Returns empty list for a hotel that does not exist in the database.
4. AGREED-BAKERIES: takes a single argument (a hotel) and returns a list bakeries that the hotel has an agreement with (e.g. `(hasanpasa istinye)`) for `(AGREED-BAKERIES 'hilton)`). Returns empty list for a hotel that does not exist in the database.

3.2 List Construction

- AGREED-HOTELS: takes a single argument (a bakery) and returns a list of hotels that have an agreement with that bakery (e.g. `(hilton)`) for `(AGREED-HOTELS 'hasanpasa)`). Returns empty list for a bakery that does not exist in the database.
- BOUGHT-BY: takes a single argument(a baked good) and returns a list of hotels that would like to buy that good (e.g. `(conrad hyatt)`) for `(BOUGHT-BY 'simit)`). Returns empty list for a baked good that does not exist in the database.

3.3 Price Calculation

- BEST-PRICE: takes a single argument (a baked good) and returns the minimum price for it. (e.g. `2`) for `(BEST-PRICE 'simit)`). Returns empty list for a baked good that does not exist in the database. Check whether the baked good is actively sold by a bakery.
- WITHIN-BUDGET: takes two arguments (a min price and a max price) and returns the list of baked goods-bakery pairs which have sale price between these arguments (inclusive) and ordered by price (e.g. `((pogaca aynali) (simit istinye) (portakalli istinye) (corek hasanpasa) (kek aynali))`) for `(WITHIN-BUDGET 3 5)`). Check whether the min price is less than or equal to the max price, show a message if it is not the case. Check whether the baked good is actively sold by a bakery.
- TOTAL-COST-GOOD takes two arguments (a hotel and a good) and returns the minimum cost of buying that baked good for the hotel. The hotel can buy the baked good from a bakery that has an agreement with the hotel and sells the baked good. The total cost is the sum of the sale price for that good and the delivery cost of the bakery (e.g. `62`) for `(TOTAL-COST-GOOD conrad simit)`). Check whether the baked good is actively sold by a bakery.
- TOTAL-COST-LIST: takes a single argument (a hotel) and returns the minimum total cost for buying all the baked goods in the hotel's list. The delivery cost is calculated separately for each baked good. This means that, even if the hotel buys two baked goods from the same bakery, the hotel will pay separate delivery costs for each baked good (e.g. `0` for `(TOTAL-COST-LIST 'marmara)`, `138` for `(TOTAL-COST-LIST 'conrad)`). Check whether the baked good is actively sold by a bakery.

3.4 Submission Details

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

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

where you write your own student ID and your own name. Submit a single file on moodle and name the file as `bakery-solution.rkt`. Your Scheme program should operate on a database as illustrated above. A complete sample database called bakerydb.rkt will be provided; you can use this database for testing your program. Do NOT include the define statements for the HOTELS, BAKERIES and GOODS 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 other auxiliary functions.

3.5 Important Notes

- The project will be done individually, do not share your specific solutions with others.
- There will be a question thread in Piazza for this project. 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 construct that causes any form of mutation (impurity).

- 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