C211 Midterm Makeup – Fall 2014

Name: _____

Student Id (last 4 digits): _____

- You may use the usual primitives and expression forms from Beginning Student; for everything else, define it.
- As a shortcut, you may write `c → e` instead [a] of `(check-expect c e)`.
- We expect data definitions to appear in your answers, unless they're given to you. We expect you to follow the design recipe on all problems. In a number of cases, defining helper functions will be useful.
- If they are given to you, you do not need to repeat the signature and purpose statements in your implementations[b]. Likewise, you do not need to repeat any test cases given to you, but you should add tests wherever appropriate.
- Some basic test taking advice: Before you start answering any problems, read *every* problem, so your brain can be thinking about the harder problems in background while you choose and knock off the easy ones in the foreground.

Problem	Points	/out of
1		/ 20
2		/ 15
3		/ 12
4		/ 9
5		/ 19
Total		/ **75**

Good luck!

[a] For example you can write `(add1 3)` → `4` instead of the traditoinal `(check-expect (add1 3) 4)`

[b] But, as the initial midterm taught us, if you are in doubt—err on the side of being too careful.

Problem 1 C211 instructors have decided to travel to Europe where there are many languages spoken in many different countries (e.g., `"Serbian"`, `"Polish"`, `"Portuguese"`, `"Greek"`, `"Hungarian"` and many others). 20 POINTS

1. Create a data definition for *Language*, and then write a data definition for a *ListOfLanguage*. Write three examples of *ListOfLanguage*.

2. Even though there are many languages in Europe, the instructors are stopping in Spain first. Even though the main language in Spain is indeed Spanish, there are many languages spoken there that are similar to Spanish but are not Spanish[1]. Define a function `is-spanish?` that determines if a given *Language* is Spanish.

[1] `"Basque"`, `"Catalan"`, `"Galician"`, `"Aragonese"`, `"Asturian"`, `"Cantabrian"`, `"Extremaduran"`, `"Fala"` and still more, some unrecognized.

3. Learning a lot of languages can be difficult. It is tough to know more than three languages, but 3 languages is more than enough languages to be able to travel successfully throughout many countries in Europe. Design a function called `enough-languages?` that determines if the number of *Language*s in a *ListOfLanguage*s is equal to or more than 3.

4. Each instructor has a *ListOfLanguage*s associated with them. For example, Mardin has the ability to speak Spanish, English, Farsi, Assyrian, and Dutch. Since the instructors will be stopping in Spain first, they will not need to remember that they know Spanish after they leave Spain. Design a function `remove-spanish` which takes a *ListOfLanguage* and produces a new *ListOfLanguage* which has Spanish removed.

Problem 2 Belgium is a relatively small country. One can drive out of it in one hour from almost anywhere, and not even half an hour from most cities. Arrived in Bruxelles, Belgium, the C211 instructors find out that their travel agency offers two types of *DayTrip*s: one that has a fixed itinerary (guided) and the other one that does not set any limits on what and when you see (unguided). 15 POINTS

1. Create a data definition and structure definition appropriate for storing information about the *DayTrip*s. In both cases the information includes the destination. The fixed itinerary day trip includes a number of hours to be traveled by train, a number of hours in the company of a local guide and a number of tokens to be used for admission to the various attractions in town. The other kind of trip includes just a number of hours of rental car. The C211 instructors are contemplating trips to Lille[2] (France), Maastricht[3] (Netherlands), Monschau[4] (Germany) and Luxembourg City (Luxembourg). Use your data and structure definitions to encode the trips to Lille, Maastricht and Monschau as described above.

[2] An hour by train (round-trip), six hours of guided tours and 20 admission tokens

[3] Three hours round-trip by train, four hours of guided sight-seeing and 15 tokens

[4] Four hours round trip by car since there is no train station in Monschau; car can be rented for a whole number of hours, usually 12 or 24.

2. Design the `total-cost` function that takes a *DayTrip* and computes the cost needed to book it. Here's the domain knowledge you need to implement this function: One hour by train is $3.25, one hour of guided assistance is $4.25 (since you usually join a group of tourists led by that guide—you meet them at the train station at a certain time) and one hour of rental car is $1.75 (you take care of gas, maps and everything else). Thirty minutes of helicopter shuttle to anywhere (this will come in useful a bit later) is $10 and an admission token is worth $2.75.

3. Design a new variant of a *DayTrip* for trips that require special clearance. The trip to Luxembourg City[5] is such a (special) trip, where you visit the vaults of the numerous banks in town. The fee for paperwork for these trips is fixed ($25) and recorded in the data definition for the trip. Adapt the data definiton of *DayTrip* and your `total-cost` function to handle such trips.

   You do not have to repeat things you wrote earlier in this problem, but if you don't, you must make clear exactly what changes you want to make to data definitions, signatures, purpose statements, function definitions, and tests.

[5] Round trip by helicopter shuttle is about ninety minutes (or one hour and a half).

[Here is some more space for the previous problem.]

Problem 3 In the City of Pamplona, one can run with the bulls if he or she dares to risk his or her life. There is no official list of all the people who have run with the bulls since there is no registration list. The mayor of Pamplona has decided that there will be a list of people who are registered to run with the bulls this year. With each person who registers, the list gets longer and longer, but the people get added to the bottom of the list. 12 POINTS

To start with, here's the data definition for this problem:

```
; A Runner is:
; -- String
; example "myadegar", "dgerman", "zhicwang"

; A ListOfRunner is one of:
; - empty
; - (cons Runner ListOfRunner)
```

1. Create a function called `register-runner` which adds someone to the last part of the list of registrants. Here are two examples:

```
(check-expect (register-runner "schackb" empty)
              (list "schackb"))
(check-expect (register-runner "samth" (list "earldean" "eeleonha"))
              (list "earldean" "eeleonha" "samth"))
```

2. The mayor of Pamplona under pressure from the opposition has agreed that he will allow those who registered first to run last in line. Using only `register-runner` defined earlier, and possibly a recursive call to itself, write a function `cambiar` that puts the elements of a list of registrants in reverse order (no, you can't use the BSL function `reverse`!).

```
(check-expect (cambiar empty) empty)
(check-expect (cambiar (list "earldean" "eeleonha" "samth"))
              (list "samth" "eeleonha" "earldean"))
```

[Here is some more space for the previous problem.]

Problem 4 Consider the following data definition and structure definition. 9 POINTS

```
;; A Country is a structure:
;;   (make-country Number Number)
(define-struct country (area population))
```

1. List the functions that are automatically created by this definition.

2. Give two examples of *Country*-ies.

3. Design the function `same-size?`, which consumes two *Country* structures and determines if they are the same size. Two country objects are said to be same size if the populations (or the areas) are within 2% of each other (i.e., one of the numbers is between 98% and 102% of the other number).

   As an example, from this point of view, the country of Belgium[6] is about the same size as Greece[7] (population) which is about the same size as England[8] (area). But Belgium and England are not the same size. As an additional example, Slovakia[9] and Austria[10] are not the same size either—whether compared with each other or with any of the other countries mentioned above.

[6] 11,787 square miles and 11.2 million people
[7] 50,949 square miles and 11.03 million people
[8] 50,346 square miles and 53.01 million people
[9] 18,933 square miles and 5.41 million people
[10] 32,377 square miles and 8.474 million people

[Here is some more space for the previous problem.]

Problem 5 The C211 instructors are now in Italy. So much to see, so little time! Since they stay in Italy only five days then need to make a *Plan*: 19 POINTS

```
;; A Plan is one of:
;;  - (make-empty-plan)
;;  - (make-trip-item Number String Plan)
(define-struct empty-plan ())
(define-struct trip-item (day destination rest-of-plan))
```

1. Sam says he wants to start the group's trip in Sicily and end in Rome (on the fifth day). Adrian insists they should spend a day visiting Naples and another one visiting Pompeii; he suggests days 2 and 3 of the trip. They can't agree on what to plan for the fourth day of the trip as Erik wants to take the group to Milan while Mardin suggests a trip to Piazza San Marco in Venice. Write down the BSL *Plan* that has the following three pairs in it: (5, Rome), (1, Sicily), (3, Pompeii).

2. Design the function `set`, which takes a *Plan*, a *Number*, and a *String*, and produces a new *Plan* that associates the given number (day) with the given string (destination). Write down the `set` call that adds (2, Naples) to the plan listed earlier.

[Here is some more space for the previous problem.]

Let `p1` be the *Plan* resulted from the successive application of `set` starting from the `empty-plan` and that contains the pairs (5, Rome), (1, Sicily), (2, Naples) and (3, Pompeii). This was the result of many hours of debate and argumentation (that maybe could have been better spent visiting). Regardless, at this point Sam asks two questions: "Do we have *anything* planned for Day 2?" and "*What* do we have planned for Day 2?"

3. Write the sequence of `set` invocations that produces `p1`, then define a function `has-key?` that takes a *Plan* and a *Number* and helps answer Sam's first question by determining if the *Plan* has the *Number* inside, as a key. Write a `check-expect` that shows the specific answer for `p1` and 2.

4. Write a function `lookup` that takes a *Plan* and a *Number* and returns a `String` and helps answer Sam's second question. Write a `check-expect` for `(lookup p1 2)`. After getting his answers Sam sets the value of Milan for Day 4 in `p1`. Write a `check-expect` for `(lookup p1 4)`. Shortly after Sam's call to `set`, Adrian also calls `(set p1 4 "Venice")`, at the request of Mardin. What does the `check-expect` for `(lookup p1 4)` look now?

[Here is some more space for the previous problem.]