CS 115 Fall 2012

Assignment 09

Due: at 4:00 PM on Monday, December 3, 2012

Assignment Guidelines:

- ✓ The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.
- ✓ Do not copy the purpose directly from the assignment description. The purpose should be written in your own words and include reference to the parameter names of your functions.
- ✓ Do not send any code files by email to your instructors or tutors. It will not be accepted by course staff as an assignment submission.
- ✓ Read the course Web page for more information on assignment policies and how to organize and submit your work.
- ✓ Download the interface file from the course Web page.
- ✓ Follow the instructions in the style guide (see the updated version posted on the course Web page). Specifically, your solutions should be placed in files `a09qY.rkt`, where `Y` is a value from 1 to 4.
- ✓ For full marks, it is not sufficient to have a correct program. Be sure to follow all the steps of the design recipe, including the definition of constants and helper functions where appropriate. LOCAL helper functions only require a contract and purpose, and must appear immediately before the function header.
- ✓ Test data for all questions will always meet the stated assumptions for consumed values.
- ✓ Read each question carefully for restrictions.
- ✓ Use only material from Modules 1 through 10 in your solution. In particular, do not use the function `reverse` or `length` with lists.

Language level: Intermediate Student.

Coverage: Module 10.

Note for questions 1, 2 and 4: Do not use recursion directly in your solution. Instead, you should make use of the built-in abstract list functions `map`, `filter` and `foldr`.

Note for all questions: All helper functions should be included in the main function using `local`.

1. Write a Scheme function `evens-odds` (using at least one abstract list function), which consumes a list of numbers `lst` and produces the difference between the sum of the odd numbers and the sum of the even numbers in the list `lst`. Some examples follow:

```
(evens-odds (list 1 2 3 4)) => -2
(evens-odds empty) => 0
(evens-odds (list 2 4 16)) => -22
(evens-odds (list 1 3 7 21 13)) => 45
```

   Note: You should use local function(s) only (in case of need).

2. Write a Scheme function `find-common` (using at least one abstract list function), which consumes two lists of numbers `lst1`, and `lst2` and produces a list of all elements of `lst1` that appear in `lst2`, with the same order of `lst1` . Some examples follow:

```
(find-common empty empty) => empty
(find-common (list 2 3 56 23 3) (list 3 2 4 56 4 14)) =>
                            (list 2 3 56 3)
(find-common (list 56 29) (list 156 329  43)) => empty
```

3. Write a Scheme function `sorted-intersection`, which consumes two lists of numbers `lst1` and `lst2`, and produces a <u>sorted</u> list (descending order) containing all numbers that are both in `lst1` and `lst2`. Some examples follow:

```
(sorted-intersection (list 2 3 3)(list 2 3)) =>
                                        (list 3 2)
(sorted-intersection(list 13 1 12 12 13 33)(list 4 12 33 4)
                    => (list 33 12)
(sorted-intersection (list 2 3 3)empty) => empty
(sorted-intersection (list 20 33 33)(list 33 20 33))=>
                                        (list 33 33 20)
```

   Note: You should use local function(s) and local constants only. You don't have to use abstract list functions. You may use built-in function `sort`, some examples follow:

```
(sort (list 6 7 2 1 3 4 0 5) <) => (list 0 1 2 3 4 5 6 7)
(sort (list 6 7 2 3 4 5 9 8) >) => (list 9 8 7 6 5 4 3 2)
```

4. Write a Scheme function `ismember?` (using at least one abstract list function) which consumes two lists of any type `lst1` and `lst2`, and produces a list of booleans, where the ith member is true if the ith member in `lst1` appears in `lst2`, otherwise false. Some examples follow:

```
(ismember? empty empty) => empty
(ismember? (list 2 5)(list 5 2 3)) => (list true true)
(ismember? (list 10 5 7 12)(list 15 7 22 10))  =>
                          (list true false true false)
(ismember? (list 10 5 10 12)(list 15 22 10)) =>
                          (list true false true false)
(ismember? (list 10 5 7 12)empty) =>
                         (list false false false false)
(ismember? (list "nice" "no" "nice" "never")
           (list "yes" "day" "nice"))=>
                         (list true false true false)
(ismember? (list 'red 'black 'white 'symbol)
           (list 'empty 'dark 'red 'navy))  =>
                        (list true false false false)
(ismember? (list 'red "black" 5 false)
           (list false 'dark "black" 15)) =>
                         (list false true false true)
```

Note: Use at least one abstract list function. You should use local function(s) only.