CS 115 Fall 2012

Assignment 08

Due: at 9:00 AM on Wednesday, November 28, 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 `a08qY.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.
- ✓ 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 9 in your solution. In particular, do not use the function `reverse` or `length` with lists.

Language level: Beginning Student with List Abbreviations.

Coverage: Module 9.

Note: We strongly recommend the careful use of templates on this assignment in particular.

The following structure and data definitions are required for the assignment. You will also need to use the data and structure definitions from the compound teachpack.

Useful structure and data definitions:

```
;; A boolean expression bexp is one of:
;; a boolean value, or
;; a boolexp, or
;; a compexp.

(define-struct boolexp (fn args))
;; A boolexp is a structure (make-boolexp f a) where
;; f is either 'and or 'or and
;; a is a bexplist

(define-struct compexp (fn nl))
;; A compexp is a structure (make-compexp f n) where
;; f is a symbol chosen from  '<, '>, and '=
;; n is a listof two numbers

;; A bexplist is either
;; empty  or
;; (cons b blist) where
;;      b is a bexp and
;;      blist is a bexplist

;; A leaf-labelled tree llt is one of the following:
;; empty
;; (cons l1 l2) where
;;      l1 is a non-empty llt and
;;      l2 is a llt
;; (cons v l) where
;;      v is a string and
;;      l is a llt

(define-struct directory (name contents))
;; A directory is a structure (make-directory n c) where
;;        n is a symbol (directory name) and
;;        c is lof (listof files)

;; A file is either
;;  a symbol (file name) or
;;  a directory (make-directory n c) where
;;        n is a symbol (directory name) and
;;        c is lof (listof files)

;; A list of files lof is either
;; empty    or
;; (cons f lofd) where
;;        f is a file and
;;        lofd is a lof
```

1. Write a Scheme function `min-elm` that consumes `elm` of type `element` and `comp1` and `comp2` of type `compound`, and produces the symbol of the `compound` that has fewer total number of atoms of type `elm`. If both, `comp1` and `comp2`, have the same total number of atoms, the function should produce 'both. Some examples follow:

```
(min-elm iron iron-sulfate iron-sulfate) => 'both
(min-elm oxygen po-four so-four)  => 'both
(min-elm oxygen po-four c123) =>  'c123
(min-elm phosphorus po-four calcium-phosphate) => 'PO4
(min-elm iron iron-sulfate po-four) => 'PO4
```

Note: This question uses the compound teachpack, which you can download from the course webpage under the Resources/DrRacket & teachpacks link, and add it to your a08q1.rkt file. You may use the constants defined in the teachpack in your testing and examples. Do not copy the compound definitions into your submitted file, as it causes our tests to fail and you will lose correctness marks.

2. Write a Scheme function `bl-eval` that consumes a boolean expression `bexp`, and produces the boolean value of `bexp`. Some examples follow:

```
(define be1 (make-boolexp 'and
(list
   (make-boolexp 'or
              (list (make-compexp '= (list 115 115 )) false))
   (make-boolexp 'or
               (list(make-compexp '< (list 15 6)) false))
)))
(define be2 (make-boolexp 'or (list true true false)))
(define be3 (make-compexp '= (list 15 15 )))
(define be4 (make-boolexp 'and
            (list (make-compexp '= (list 4 4))
                  true
                  (make-compexp '< (list 5 6)))))
(define be5 (make-boolexp 'and (list be1 be2 be3)))
(define be6 (make-compexp '> (list 15 15 )))
(define be7 (make-boolexp 'and (list be3 be6 )))

(bl-eval true) => true
(bl-eval be2)  => true
(bl-eval be4)  => true
(bl-eval be5)  => false
(bl-eval be6)  => false
(bl-eval be7)  => false
```

3. Write a Scheme function `palindrome-strings` that consumes a leaf-labelled tree `lt`, where all labels are strings, and produces a list of the strings in `lt` that are palindromes. Any order is acceptable. (a string is a palindrome if its reverse is identical to it, for example: “wow” “abcddcba”). Some examples follow:

```
(define myllt '(
                 ("hat" ("cat" "dog" "wow")
                        ("abcba" "Wow" "nice")
                        ("halah" "lol"))
                 ("was saw" "end")))
(define myllt2 '(
                 ("hat" ("cat" "dog" "woww")
                        ("abcbda" "nice")
                        ("hala" "lolo"))
                 ("was see" "end")))
(define myllt3 '( "aaa"
                  ("hat" ("cat" "dog" "woww") ("hala" "baab"))
                  ("was see" "end")))
(define res1 '("wow" "abcba" "halah" "lol" "was saw"))
(palindrome-strings myllt)  => res1
(palindrome-strings myllt2) => empty
(palindrome-strings myllt3) => (list "aaa" "baab")
```

Note: Do not use the function `reverse` or any user defined function that reverses a list. Same for `length` (with list).

4. Write a Scheme function `find-file-lof?` that consumes a symbol `fname` and a listof files, `flist`, and produces `true` if a file named `fname` exists in the `flist`, otherwise, `false`. Some examples follow:

```
(define dir2 (make-directory 'd2 (list 'f6 'f7)))
(define dir1 (make-directory 'd1 (list 'f4 'f5)))
(define dir12 (make-directory 'd12 (list 'f16 'f17)))
(define dir5 (make-directory 'd5 (list 'f14 'f55 dir12 )))
(define dir1b (make-directory 'd1 (list 'f4 'MMM)))
(define dir5b (make-directory 'd5 (list 'f14 'f55
         (make-directory 'MMM (list 'f16 'f17)))))
(define mylof (list 'f1 'f2 dir1 dir2 'f8))
(define mylof2 (list 'f1 'f2 dir1 dir2 'f8 dir5))
(define mylof3 (list dir1b dir2 'f1 'f2 'f8 dir5b))
(find-file-lof? 'f7 mylof)   => true
(find-file-lof? 'fff mylof)  => false
(find-file-lof? 'f14 mylof2) => true
(find-file-lof? 'fff mylof2) => false
(find-file-lof? 'f16 mylof3) => true
(find-file-lof? 'f55 mylof3) => true
(find-file-lof? 'fff mylof3) => false
(find-file-lof? 'd2 mylof3)  => false
```