CS 115 – Dubai – Fall 2012

Assignment 08

Due: Thursday, November 29, 2012 at 11:59 p.m.

- For this assignment you will need to download the `compound.rkt` teachpack found under the Resources tab on the course webpage. It contains structure definitions and constants that can be used for testing compounds.
- Do **not** paste material from the teachpack into your assignment files, or auto-testing will fail (because constants or structures will be defined twice).
- On this assignment in particular, it is very important to use templates to guide you. The solutions can be completed with relatively short helper functions. If you find yourself creating single, very long and complicated functions, revisit the data definitions to help with the design of your solutions.
- Remember that in the case of mutual recursion, you only need to provide one set of examples and test cases for the collection of functions you write based on the data definitions and templates.

- Do **not** use reverse or member in any of your solutions.
- You may want to include defined constants to help reduce the writing for the examples and tests.
- For this and all subsequent assignments, you are expected to use the design recipe when writing functions from scratch. Be sure to follow all the steps of the design recipe, including the definition of constants and helper functions that include the design recipe, where appropriate.
- Do not copy the purpose directly from the assignment description. The purpose should be written in your own words and include references to the parameter names of your functions.
- 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 send any code files by email to any course staff. It will not be accepted by course staff as an assignment submission. Course staff will not debug code emailed to them.
- Test data for all questions will always meet the stated assumptions for consumed values.
- Read each question carefully for restrictions.
- Read the course Web page for more information on assignment policies and how to organize and submit your work. Follow the instructions in the style guide. Specifically, your solutions should be placed in files a08qY.rkt, where Y is a value from 1 to 3.
- Download the interface file from the course Web page.

Language level: Beginning Student with List Abbreviations
Coverage: Module 8

Useful structure and data definitions:

```
define-struct ae (fn args))
;; A arithmetic expression (aexp) is either
;;   * a number, or
;;   * a structure (make-ae f alist), where
;;       * f is a symbol in the set '* or '+
;;       * alist is an aexplist.

;; An aexplist is either
;;   * empty or
;;   * (cons a alist), where
;;     * a is an aexp and
;;     * alist is an aexplist.

;; A leaf-labelled tree (llt) is one of the following:
;;   empty,
;;   (cons l1 l2) where l1 is a non-empty llt and l2 is an llt, or
;;   (cons v l) where v is an integer an l is an llt.
```

Assignment 08

Due: Thursday, November 29, 2012 at 11:59 p.m.

```
;; A compound is a structure (make-compound n l), where
;;   n is a symbol and
;;   l is a list of parts

;; A part is a structure (make-part s e), where
;;   s is an integer representing the number of units of e and
;;   e is an element or compound.

;; A list of parts is either
;;   empty or
;;   (cons p lop), where
;;    p is a part and
;;    lop is a list of parts.

;; An element is a structure (make-element n m) where
;;   n is a symbol and
;;   m is a number (the molar mass, that is the mass of one mole of the substance, 6.02 x 10^23 atoms).
```

1. Recall the substitution rules in Scheme from Module 1 that are applied when completing a full trace of a Scheme expression. Assume that you are tracing an arithmetic expression that only contains non-negative numbers, and the mathematical functions `+` and `*`. Here is a sample trace:

```
(+ 5 (* (+ 1 2 3) 2 (* 4 1)) 7)
=> (+ 5 (* 6 2 (* 4 1)) 7)
=> (+ 5 (* 6 2 4) 7)
=> (+ 5 48 7)
=> 60
```

   The substitution rules that are used in this kind of trace are:

   - A value (such as a number) cannot be further simplified.
   - For a built-in function application, use mathematics rules.
   - First evaluate the arguments, and then apply the function to the resulting values.
   - When there is a choice among two or more substitutions, we take the leftmost one.

   Each of the lines in the trace above is an example of arithmetic expressions in Scheme. Any arithmetic expression in Scheme can be represented by an `aexp`. Write a function called `step` that consumes a Scheme expression (`ex`) in the form of an `aexp` and produces an `aexp` that represents the next step in a trace of `ex`. If `ex` is a number, then there are no more steps in the trace. In this case the function produces the symbol `'Done`. Notice in the example that simplifying an argument like `(+ 1 2 3)`, where all of the arguments following the mathematical operator are numbers, is done in just one step. For example, if the `ex1` is an `aexp` representing `(+ 5 (* (+ 1 2 3) 2 (* 4 1)) 7)` then `(step ex1)` produces an `aexp` representing the Scheme expression `(+ 5 (* 6 2 (* 4 1)) 7)`.

Assignment 08

Due: Thursday, November 29, 2012 at 11:59 p.m.

2. Write a function called `longest-path` that consumes a leaf-labelled tree (`llt`) called `tree`. The function produces the length of the longest path from the root of `tree` to any of its leaves. The length of the path from the root of a leaf-labelled tree to a leaf in the tree is equal to the number of branches between the root and the leaf node in that tree. In the example of a leaf-labelled tree on slide 30 in Module 9, the longest path is 3, since the length of the paths to the leaves containing the values `2`, `3`, `6`, `7`, and `8` are all length 3 and there are no other leaves that have a longer path.

   Notes:
   - The longest path of in an empty tree is 0.
   - It is not possible for a leaf-labelled tree to have a single node. The smallest non-empty leaf-labeled tree has at least two nodes – a root and one leaf.

3. Write a function `count-atoms` that consumes a `compound` (`c`) and a symbol (`el`) and produces the number of atoms in `c` where the name of the element matches `el`. For example `(count-atoms calcium-phosphate 'O)` produces `8` (because there are 2 `'PO4 compounds` which have 4 `'O` atoms in each), and `(count-atoms calcium-phosphate 'C)` produces `0`. Note that an element is composed of one atom.

   Add the `compound` teachpack to your solution. However, do not copy any definitions from the teachpack (there will be errors if you do). You may use the data defined in the teachpack for testing.