CS 115 – Dubai – Fall 2012

Assignment 07

Due: Thursday, November 22, 2012 at 11:59 p.m.

- For this assignment you will need to download the `taxon.rkt` teachpack. It contains structure definitions and constants that can be used for testing evolution trees. You can find the teachpack under the Resources tab on the course webpage.
- Do **not** paste material from the teachpack into your assignment files, or auto-testing will fail (because constants will be defined twice). You must not define `t-ancient` or `t-modern` in your files either, since they are defined in the teachpack.

- 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 a07qY.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 bae (fn arg1 arg2))
;; A binary arithmetic expression (binexp) is either
;;   * a number, or
;;   * a structure (make-bae f a1 a2), where
;;       - f is a symbol in the set '*, '+, '/, '-
;;       - a1 is a binexp
;;       - a2 is a binexp

(define-struct t-modern (name pop)
(define-struct t-ancient (name age left right)
;; A taxon is either a t-modern or a t-ancient.
;; A t-modern is a structure, (make-t-modern n p), where
;;   n is a string and
;;   p is a number.

;; A t-ancient is a structure (make-t-ancient n a l r), where
;;   n is a string
;;   a is a number, and
;;   l and r are taxons.
```

CS 115 – Dubai – Fall 2012

Assignment 07

Due: Thursday, November 22, 2012 at 11:59 p.m.

```
(define-struct node (key left right))
;; A binary search tree (bst) is either
;;    empty or
;;    a structure (make-node k lft rgt), where
;;        k is a number,
;;        lft is a BST, and
;;        rgt is a BST.
;; In addition, the keys in lft are all less than k
;; and the keys in rgt are all greater than or equal to k.
```

1. Write a function called `scheme-ex` that consumes a `binexp` and produces a string that represents the corresponding Scheme expression. There should be a single space between the function and the first argument, and a single space between the first argument and the second argument, but no spaces after an open parenthesis or before a close parenthesis. For example, the last tree on slide 8 of Module 8 will produce `"(/ (+ (* 2 6) (* 5 2)) (- 5 3))"`. In order to avoid problems with different representations of numbers, you may restrict your numbers to integers. You may find it helpful to use the built-in functions `number->string` and `symbol->string`.

2. Write a function called `parent-of` that consumes a `taxon` (`tree`) and a string (`s-name`). The function will return the `name` of the `t-ancient` that is the parent of a `t-modern` whose name matches `s-name`. For example, if you use the sample data from the `taxon` teachpack, `(parent-of animal "Homo Sapiens")` produces `"Primate"`, because `"Homo Sapiens"` is the name in the `t-modern` structure `human`, and `"primate"` is the name of the `t-ancient` structure that is the parent of `human` in the tree that has `animal` as its root. If a `t-modern` with the name `s-name` does not exist in tree or if `s-name` does not have a parent, then the function should produce `false`.

   Add the `taxon` 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.

3. Recall that one of the properties of a binary search tree is that if you do an in order traversal (visit the left subtree recursively, visit the root, visit the right subtree recursively), then you will actually visit the keys in ascending order. You can take advantage of this property and write a function that will sort a list.

   Write a function `bst-sort` that consumes a list of numbers and produces a sorted list of those numbers. To sort the list, first add the numbers one at a time to a binary search tree. Then do an in order traversal of that tree and produce the list of the numbers you visit in order. The list produced will be in order.

   Notes:
   - You will need helper functions to build the tree and traverse the tree.
   - There may be duplicate values in the list that is to be sorted.
   - The list consumed, may be empty.

Assignment 07

Due: Thursday, November 22, 2012 at 11:59 p.m.

You will be using a slightly modified version of the data and structure definitions of `bst` that as been provided at the beginning of this assignment. For example, `(bst-sort (list 2 3 1))` would first produce the BST `(make-node 2 (make-node 1 empty empty) (make-node 3 empty empty))` and then produce `(list 1 2 3)`. **A solution that simply sorts the list, without using this technique, will get 0 correctness marks.**