**CSSE 304 Assignment #9**

Practice with define-syntax and define-datatype

This is an individual assignment. No mutation is allowed for problems 2 and 3.

1. (30 points)

(a) Extend the definition of my-let produced in class to include the syntax for named let. This should be translated into the equivalent letrec expression.

> **(my-let fact ([n 5]) (if (zero? n) 1 (\* n (fact (- n 1)))))**

120

(b) Suppose that or was not part of the Scheme language. Show how we could add it by using define-syntax to define my-or, similar to my-and that we defined in class. This may be a little bit trickier than my-and; the trouble comes if some of the expressions have side-effects; you want to make sure that no expression gets evaluated twice. In general, you’re my-or should behave just like Scheme's or.

> **(begin (define a #t)**

**(define x** **(my-or #f (begin (set! a (not a)) a) #t (set! a (not a))))**

**(list a x))**

(#f #t)

(c) Use define-syntax to define += , with behavior like += in other languages.

> **(begin (define r 4) (define y (+ 6 (+= r 3))) (list r y))**

(7 13)

(d) As you know, (begin e1 … en) evaluates the expressions e1 … en in order, returning the value of the last expression. It is sometimes useful to have a mechanism for evaluating a number of expressions sequentially and returning the value of the *first* expression. I call that syntax return-first. Use define-syntax to define return-first.

**> (define a 3) (begin a (set! a (+ 1 a)) a)**

4

> **(define a 3) (return-first a (set! a (+ 1 a)) a)**

3

2. (5 points) bintree-to-list. EOPL Exercise 2.24, page 50. This is a simple introduction to using cases and the bintree datatype (bintree definition is defined on page 50). See notes below on using define-datatype and bintree.

3. (25 points) max-interior. EOPL Exercise 2.25, page 50. This one is trickier than it looks at first! You may not use mutation. You may not traverse any subtree twice (such as by calling leaf-sum on each node). You may not create an additional data structure that you then traverse to get the answer. Think about how to return enough info from the recursive calls to do this without another traversal.

**Code to use for #2 and #3:** Copy this code to the beginning of your file.

; Binary trees using define-datatype

(load "chez-init.ss")

;; from EOPL3, page 50

(define-datatype bintree bintree?

(leaf-node

(num integer?))

(interior-node

(key symbol?)

(left-tree bintree?)

(right-tree bintree?)))

**TO USE DEFINE-DATATYPE with petite Chez Scheme**

Download [chez-init.zip](http://www.rose-hulman.edu/class/cs/csse304/201030/Resources/chez-init.zip) \* and extract the files to the folder in which you will be doing the assignment.  
Than place this code at the beginning of your program file: (load "chez-init.ss")

\*http://www.rose-hulman.edu/class/cs/csse304/201030/Resources/chez-init.zip

**TO USE DEFINE-DATATYPE with Dr. Scheme**

In Dr. Scheme’s Languages Menu, select the Essentials of Programming Languages language.

**TO USE DEFINE-DATATYPE with the Grading Program**

The chez-init files are automatically loaded by the grading program, so you should not have   
to do anything special.