CSE 341 — General Programming Language Concepts — Mini Exercises — Answers

1. Consider the following example in Ruby.

   ```
   def test k
     k = k+5
     print k
   end

   n = 0
   test n
   print n
   ```

   (a) What is the output in normal Ruby?
   5
   0

   (b) What would the output be if k were passed by reference?
   5
   5

2. Here is a Racket example.

   ```
   (define a 3)

   (define (test x)
     (printf "starting test - x = ~a\n" x)
     (set! a (+ a 1))
     (printf "after first set! - x = ~a\n" x)
     (set! a (+ a 1))
     (printf "leaving test - x = ~a\n" x))

   (test (+ a 10))
   ```

   (a) What is the output in normal Racket?

   ```
   starting test - x = 13
   after first set! - x = 13
   leaving test - x = 13
   ```

   (b) What would the output be if x were passed by reference? **The same!**

   (c) What would the output be if x were passed by name?

   ```
   starting test - x = 13
   after first set! - x = 14
   leaving test - x = 15
   ```

   (d) Rewrite the example to simulate call by name by passing a lambda.

```
    (define a 3)
    (define (test x)
      (printf "starting test - x evaluated = ~a\n" (x))
      (set! a (+ a 1))
      (printf "after first set! - x evaluated = ~a\n" (x))
      (set! a (+ a 1))
      (printf "leaving test -x evaluated = ~a\n" (x)))

      (test (lambda () (+ a 10)))
```

3. True or false?

   (a) Haskell is statically typed if the programmer includes a type declaration for all functions; otherwise it is dynamically typed. **False**.

   (b) Java is type safe. **True**.

   (c) Each of the following Haskell expression gives a compile-time type error, since `tail` is being provided a value of the incorrect type:

   ```
   tail []
   tail (1,2,3)
   ```

   False. (Only the second gives a type error; the first one gives a runtime error.)

4. What happens when you try the following Haskell program?

   ```
   x :: Float
   y :: Double
   x = 3
   y = 4
   z = x+y
   ```

   You get a type error, since + doesn't work with two different types (Double and Float). No coercion in Haskell, not even Float to Double. But note that Haskell isn't troubled by `x = 3`! That's ok because 3 has type `(Num t) => t`.