Quiz 12a

1. (1+2+2 points) Now that we have the lazy evaluator, why should we bother keeping if as a special form? We could just define it as a procedure, **and remove** the if-exp? clause in mc-eval:

;; Somewhere in mc-eval, the if-exp? clause is removed

;; In STk

(define (new-if test true-case false-case)

(if (force-it test) true-case false-case))

(set! primitive-procedures (cons (list ‘if new-if) primitive-procedures))

Here is a list of potential problems with this definition for if. For each one, state with justification whether it really is a problem.

* 1. Lazily evaluating the true-case and false-case of the if is not sufficient – there are cases in which such an if would not behave as intended. (No need for justification here.)
  2. This version of if is not actually lazy – both the true-case and the false-case will always be evaluated.
  3. This will cause an infinite loop – even though we named the procedure new-if, we added a binding to primitive-procedures in which its name was if. So, when we try to evaluate an if, we have to evaluate (if (force-it test) …), which will lead to infinite recursion.

1. (1+1+1+1+1 points) Recall that one place where we call force-it is when we print a value in the REPL. Suppose we forgot to make this change, so that we just print whatever mc-eval returns without forcing it. (Assume we use the memoized version of the lazy evaluator for this question.)
   1. For each of the following, what would be printed? For a thunk, write “thunk:” followed by the thunk expression. If it prints a delayed (+ 2 5), you should write thunk: (+ 2 5). If it causes an error, write Error.

(define (mystery x y)

(x 2 (begin (set! y 10)

4))

y)

(mystery (lambda (a b)

(+ a b))

(+ 4 9))

(mystery (lambda (a b)

(set! b 3)

(+ a b))

(+ 4 9))

* 1. For each of the following, say whether the **new** lazy evaluator would evaluate and print an answer faster than the regular metacircular evaluator.

((lambda (x y) (+ x y)) (\* 4 4) (\* 9 9))

Faster Not faster

((lambda (x y) (\* 2 x)) (\* 4 4) (\* 9 9))

Faster Not faster

(define x 4)

(define (foo a b)

(set! x (+ a 2))

b)

(foo (\* 6 9) (\* 8 3))

Faster Not faster