CSE 341 — More Scheme Discussion Questions

1. Using \texttt{map} and \texttt{lambda}, define a function \texttt{averages} that accepts two lists of numbers, and returns a list of the average of each pair. For example:

\begin{verbatim}
(averages '(1 2 3) '(11 12 13)) => (6 7 8)
\end{verbatim}

2. What is the value of \texttt{x} and \texttt{y} after evaluating the following expressions?

(a) (define \texttt{x }'(1 2 3 4))
   (define \texttt{y x})
   (set-car! (cdr \texttt{x}) 100)

(b) (define \texttt{x }'(1 2 3 4))
   (define \texttt{y x})
   (set! \texttt{x '}(100 200))

3. Aloysius Q. Hacker, 341 student, is puzzled by the following code.

\begin{verbatim}
(define incr)
(define get)

(let ((n 0))
  (set! incr (lambda (i) (set! n (+ n i))))
  (set! get (lambda () n)))
\end{verbatim}

Aloysius is unsure how the functions \texttt{incr} and \texttt{get} can possibly work... if \texttt{n} is in a stack frame, why do \texttt{incr} and \texttt{get} still work correctly even though we’re done with evaluating the \texttt{let}?

Is there something special about \texttt{let} at the top level? So Aloysius tries an experiment:

\begin{verbatim}
(define newincr)
(define newget)

(define (test k)
  (let ((n 0))
    (set! newincr (lambda (i) (set! n (+ n i k))))
    (set! newget (lambda () n))))
\end{verbatim}

Then he evaluates \texttt{(test 100)}. This time the \texttt{let} is embedded in a function, and so (Aloysius reasons) certainly its stack frame will go away when \texttt{test} returns.

What is the result when Aloysius evaluates each of the following expressions in turn?

\begin{verbatim}
(newget)
(newinc 10)
(newget)
\end{verbatim}

Explain (or at least make some reasonable hypotheses).

4. Define a tail-recursive version of “map” for 1-argument functions. (Avoid side effects if possible, but use them if necessary.)