UNIT 1 FINAL

1. **(3 points)  Lambda and HOFs**

   Define a function `compose-many` that takes in a list of one-argument functions and returns another one-argument function that applies each function in sequence. If our list is `(f g h)`, where `f`, `g`, and `h` are all functions that take in one argument, `compose-many` should return the equivalent of `(lambda (x) (f (g (h x))))`.

   Notice that `compose-many` takes in a list, not a sentence, as its argument! You may not use recursion for this problem–just use higher order functions, like `map`, `filter`, and `accumulate`.

   Hint: you can use the function `compose` from the homework, which takes in two functions, `f` and `g`, and returns `(lambda (x) (f (g x)))`

   ```scheme
   STK>((compose square square) 5)
   625
   STk>(define (add-1 x) (+ x 1))
   STk>(define add-4 (compose-many (list add-1 add-1 add-1 add-1)))
   STk>(add-4 10)
   14
   STk>(define add-16 (compose-many (list add-4 add-4 add-4 add-4)))
   STk>(add-16 4)
   20
   ```

2. **(2 points)  Orders of Growth** What are the runtimes of the following procedures in big Theta notation:

   ```scheme
   (define (mystery1 n)
     (if (= n 0)
         1
         (* (mystery (- n 1)) (mystery (- n 1))))
   
   (define (mystery2 sent)
     (if (= 0 (remainder (count sent) 4))
         sent
         (mystery2 (bf sent)))
   ```
3. (5 points) Anagrams

(a) (2 pt) Write the function \textit{remove-once}, which takes in a word and a letter, and returns that word with the first occurrence of the letter removed.

\begin{verbatim}
STk>(remove 'a 'aardvark)
ardvark
STk>(remove 'f 'bar)
bar
\end{verbatim}

(b) (3 pt) Write the function \textit{anagram}, which takes in two words and returns \texttt{true} if the words are anagrams of each other, and \texttt{false} otherwise. Two words are anagrams if they have the same length and are made up of the same letters.

Hint: you might find your solution from part (a) useful. You may assume that it works even if it doesn't.

\begin{verbatim}
STk>(anagram? 'foo 'foofoofoo)
#f
STk>(anagram? 'listen 'silent)
#t
STk>(anagram? 'yes 'no)
#f
STk>(anagram? 'foobar 'foabor)
#t
STk>(anagram? "" 'no)
#f
STk>(anagram? 'no "")
#f
\end{verbatim}