CMPT 383
Quiz #8
December 1, 2005

1) Imagine a language of expressions for representing integers defined by the syntax rules
   i) `zero` is an expression
   ii) If `e` is an expression, then so are `succ(e)` and `pred(e)`.
   iii) If e_1 and e_2 are expression, then so is `add(`e_1`,`e_2`)`.

   An evaluator reduces expressions in this language by applying the following rules repeatedly until no longer possible:

   `succ(pred(e)) = e`
   `pred(succ(e)) = e`
   `add(zero,`e_2`) =` e_2
   `add(succ(`e_1`),`e_2`) = succ(add(`e_1`,`e_2`))`
   `add(pred(`e_1`),`e_2`) = pred(add(`e_1`,`e_2`))`

   Simplify the expression `add(succ(pred(zero)),zero)` using
   a) Innermost reduction sequence.
   b) Outermost reduction sequence.
   c) Can the expression be simplified using another reduction sequence?

2) Which of the following equations are *true* and which are *false*?
   a) `[]:xs = xs`
   b) `[]:xs =[[],xs]`
   c) `xs:[] = xs`
   d) `xs:[] = [xs]`
   e) `xs:xs = [xs,xs]`
   f) `[[]] ++ xs = xs`
   g) `[[]] ++  xs = [xs]`
   h) `[[]] ++ xs = [[],xs]`
   i) `[[]] ++ [xs] = [[],xs]`
   j) `[xs] ++ []= [xs]`
   k) `[xs] ++ [xs] = [xs,xs]`

3) Define a data type `Direction` whose values describe the four major points of the compass, and define a function `reverse` for reversing direction.

4) Evaluate `map (map square) [[1,2],[3,4,5]]`

5) The Fibonacci numbers f_0, f_1, … are defined by the rule that $f_0=0$, $f_1=1$ and $f_{n+2}=f_n+f_{n+1}$ for all `n>=0`. Give a definition of the function `fib` that takes an integer `n` and returns f_n.

6) Using pattern matching, define a function `rev2` that reverses all lists of length `2`, but leaves others unchanged. Ensure that the patterns are exhaustive and disjoint.

7) Define a function `nextlet` that takes a letter of the alphabet and returns the letter coming after it. Assume that letter `'A'` follows `'Z'`.

8) Explain informally, but clearly and fully, what the function `assert_length`, defined below, does

```
assert_length           :: Integer → [a] → [a]
assert_length 0 xs      = []
assert_length (n+1) xs  = (head xs) : assert_length n (tail xs)
```