Chapter 15

EXERCISE 1: Identify the redexes in the following expressions, and determine whether each redex is innermost, outermost, neither, or both:

```
1 + (2*3)
(1+2) * (2+3)
fst (1+2, 2+3)
(\x -> 1 + x) (2*3)
```

The expression `1 + (2*3)` has one redex `2*3` which is both innermost and outermost.

The expression `(1+2) * (2+3)` has two redexes, `1+2` and `2+3`. The redex `1+2` is both innermost and outermost.

The expression `fst (1+2, 2+3)` has three redexes, `1+2` is innermost and the whole expression is outermost.

The expression `(\x -> 1 + x) (2*3)` has two redexes, `2*3` which is innermost and the whole expression which is outermost. The expression `1 + x` inside the lambda is not a redex since we do not reduce under lambdas.

EXERCISE 2: Show why outermost evaluation is preferable to innermost for the purposes of evaluating the expression `fst (1+2,2+3)`.

Innermost evaluation results in:

$$\begin{aligned} \texttt{fst}(1+2, 2+3) &= \texttt{fst}(3, 2+3) \\ &= \texttt{fst}(3, 5) \\ &= 3 \end{aligned}$$

Outermost evaluation results in:

$$\begin{aligned} \texttt{fst}(1+2, 2+3) &= 1+2 \\ &= 3 \end{aligned}$$

So outermost evaluation does not evaluate `2+3` which is discarded anyway by `fst`.

EXERCISE 3: Given the definition `mult = \x -> (\y -> x * y)`, show how the evaluation of `mult 3 4` can be broken down into four separate steps.

```
mult 3 4 = (\x -> (\y -> x * y)) 3 4
         = (\y -> 3 * y) 4
         = 3 * 4
         = 12
```

EXERCISE 4: Using a list comprehension, define an expression `fibs :: [Integer]` that generates the infinite sequence of Fibonacci numbers

$$0, 1, 2, 3, 5, 8, 13, 21, 34, \cdots$$

using the following simple procedure:

1. the first two numbers are 0 and 1;
2. the next is the sum of the previous two;
3. return to the second step.

Hint: make use of the library functions `zip` and `tail`. Note that numbers in the Fibonacci sequence quickly become large, hence the use of the type `Integer` of arbitrary-precision integers above.

```
fibs :: [Integer]
fibs = [0, 1] ++ zipWith (+) fibs (tail fibs)
```