Functional Programming Skills
Assignment 1

Arthur Nunes-Harwitt

Each question is worth 10 points.

Explicitly write the type of each function.
Following each function there should be commented out expressions to test the function.

1. Write a function `second` that has input `lst`, where `lst` is a list. It should return the second element of the list. Do not worry about error checking. The time complexity should be $\mathcal{O}(1)$.
   Example:
   `second [4,5,6]` $\rightsquigarrow$ `5`

2. Write a predicate (i.e., a function that returns true or false) `singleton` that has input `lst`, where `lst` is a list. It should return `True` if the list has exactly one element, and `False` otherwise. The time complexity should be $\mathcal{O}(1)$.
   Examples:
   `singleton []` $\rightsquigarrow$ `False`
   `singleton [5]` $\rightsquigarrow$ `True`
   `singleton [5,6]` $\rightsquigarrow$ `False`

3. Write a function `index` that has inputs `x` and `lst`, where `lst` is a list of elements of the same type as `x`; the type of `x` should have equality. (Use the `Eq` type class.) It should return `Just` n, where n is the zero based location of the first occurrence of `x` in `lst`, or `Nothing` if there is no occurrence. The time complexity should be $\mathcal{O}(n)$.
   Example:
   `index 'x' "qrsxyz"` $\rightsquigarrow$ `Just 3`
   `index 'x' "qrsyz"` $\rightsquigarrow$ `Nothing`

4. Consider the following function.

   ```
   evenSquares :: [Integer] -> [Integer]
   evenSquares lst = [x*x | x <- lst, even x]
   ```

   Example:
   `evenSquares [1 .. 10]` $\rightsquigarrow$ `[4,16,36,64,100]`

   This function is implemented using a list comprehension. Write a function `evenSquares'` that does the same thing, but does *not* use list comprehensions. Use `map` and `filter` instead.

5. Write a function `insertionSort` that has input `lst`, where `lst` is a list of elements that can be compared. (Use the `Ord` type class.) It should return a list of the same length and with the same elements as `lst` but they should be in ascending order. The insertion-sort algorithm is structurally recursive: sort the tail of the list and then insert the head of the list in its proper place. Write `insertionSort` and its helper function `insert` using *explicit recursion*. The time complexity should be $\mathcal{O}(n^2)$.
   Example:
   `insertionSort [5,1,4,3,2,6,5]` $\leadsto$ `[1,2,3,4,5,5,6]`

6. Implement insertion-sort again. Write `insertionSortH` using `foldr` instead of explicit recursion. (You should make use of `insert` from the previous question.)

7. Write a function `perm` that has input `lst`. It should return a list of all the permutations of `lst`. (HINT: You will need at least one helper function.)
   Examples:
   `perm [2,3,4]`$\leadsto$`[[2,3,4],[3,2,4],[3,4,2],[2,4,3],[4,2,3],[4,3,2]]`
   `perm "abc"`$\leadsto$`["abc","bac","bca","acb","cab","cba"]`

8. This problem has several parts.

   (a) Define a new data structure `Peano` to represent numbers in unary. It consists of two choices: `Zero` or `S`. The choice `S` has one part of type `Peano`.

   (b) Use structural recursion to define the function `add`, which implements addition for Peano numbers. (You may *not* use conversion functions in your implementation.)
   Example:
   `add (S (S Zero)) (S (S (S Zero)))`$\leadsto$`S (S (S (S (S Zero))))`

   (c) Use structural recursion to define the function `mult`, which implements multiplication for Peano numbers. (You may *not* use conversion functions in your implementation.)
   Example:
   `mult (S (S Zero)) (S (S (S Zero)))`$\leadsto$`S (S (S (S (S (S Zero)))))`

   (d) Using pattern matching and the multiplication operator defined above, write the factorial function `fact` for Peano numbers. (You may *not* use conversion functions in your implementation.)
   Example:
   `fact (S (S (S Zero)))`$\leadsto$`S (S (S (S (S (S Zero)))))`

Graduate Problems/Undergraduate Extra Credit

1. (a) Define a function `meaning` specified as follows.

$$\begin{aligned} \llbracket \mathsf{Zero} \rrbracket &= \lambda s.\lambda z.z \\ \llbracket (\mathsf{S}\ n) \rrbracket &= \lambda s.\lambda z.(s\ (\llbracket n \rrbracket\ s\ z)) \end{aligned}$$

   (b) Use `meaning` to write a function `fromPeano` that converts Peano numbers to Haskell integers.