Programming Assignment 4

Haskell (20 Points)

Don’t show your code to anyone in the class. Don’t read anyone else’s code. Do not google to find complete solutions to the exercises (as that involves reading someone else’s code). Don’t discuss the details of your code with anyone except the tutors or your instructor.

Rather than give you one large program to do, you are being asked to do a series of short functions so that you have experience with a variety of features. Comments begin with two hyphens and continue to the end of the line. Before each function, insert comments which illustrate that the function works, i.e. show a few calls to the function and the output produced.

1. Write isPrime :: Integer -> Bool which determines whether a given integer is prime.

2. Define primes :: [Integer] which returns the list of all primes. This is an infinite list, but because Haskell uses lazy evaluation, we will only generate as much of the list as we need. A call such as take 5 primes would return the first five primes.

3. Sets can be implemented using lists, provided that Ord and Eq operations are available that implement a total order on potential set elements. Show how a polymorphic set ADT based on lists can be implemented using Haskell polymorphic types. Demonstrate that the same code works for sets of Int or sets of Color. Color is defined as an enumerated type of your favorite colors.
   ```haskell
   data Color = Red | Blue | Green | Yellow | Purple | Orange | Pink
   deriving (Show, Eq, Ord)
   ```
   Your abstraction should include the following operations:
   - member a function to check whether an element is a member of a set,
   - emptySet, a representation of an empty set, and
   - setAdd, a function to add an element into a set (note, we don't allow duplicate elements in a set)
   - setUnion, a function to combine the elements of two sets to produce a third set (note, we don't allow duplicate elements in a set)

4. Use map to define the following function: allCaps which converts a string to all uppercase. (Hint: the toUpper function is built into the Char library. To get access to it, use import Char)

   ```haskell
   import Char
   ```
allCaps "this is a GREAT day" yields "THIS IS A GREAT DAY"

5. Use map to define the following function: begList which takes a non-negative integer and produces a list of lists of integers smaller than the argument. For example,

\[
\text{begList 8} = \left[ \left[ \right], \left[ 1 \right], \left[ 1, 2 \right], \left[ 1, 2, 3 \right], \left[ 1, 2, 3, 4 \right], \left[ 1, 2, 3, 4, 5 \right], \left[ 1, 2, 3, 4, 5, 6 \right] \right]
\]

6. Define summation to take a lower bound, an upper bound, and a function. Summation will calculate the sum of the values of the function applied to each value from lower to upper. Try it with a lambda expression. For example

\[
\text{summation 0 10 (\( x \rightarrow x \times x \))} = 385: \text{Int}
\]

\[
\text{summation 5 10 (^2) -- notice the use of a curried function ^} = 355
\]

7. Write a function \( \text{prods :: Int} \rightarrow \left[ \text{Int} \right] \) such that \( \text{prods n} \) generates the infinite list of multiples of \( n \) starting with \( n^2 \).

\[
\text{prods 3} = [9, 12, 15, 18, 21, 24, \ldots]
\]

\[
\text{prods 4} = [16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, \ldots]
\]

8. The parameter passing mechanism in Haskell is pass-by-need. With this mechanism, an argument expression is only evaluated when the value is actually needed, (that is, when the corresponding parameter is accessed). This is similar to pass-by-name, except that the expression is only evaluated the first time the parameter is accessed; the result of the first evaluation is stored as the value of the parameter for the remainder of the function activation. Show an example of a function that cannot be programmed in languages using pass-by-value or pass-by-reference, but can be implemented using pass-by-need.