G6021: Comparative Programming

Exercise Sheet 10

1 Bits of past exam questions

The following questions are parts of past exam questions (or very similar to them). You would have to answer these on paper, but for this exercise sheet, try to test your answers with the Haskell interpreter in the labs.

1. Give the output for each of the following Haskell expressions. Include a brief (1 or 2 sentences) explanation to justify your answers.

   (a) `[x | x<-[1..], x<6]`

   (b) `map (\(x,y) -> x*y) [(1,3),(4,5),(1,9)]`

   (c) `foldr (\x -> \y -> 1+y) 0 "testing"`

   (d) `f [3,4,1,2] []`, where `f` is defined as:

   ```
   f [] acc = acc
   f (h:t) acc = f [x | x <- t, x<=h]
                    (h:f [x | x <- t, x>h] acc)
   ```

   Answer:
   If incorrect answers are given, but some correct working out is shown, then partial marks will given.

   ```
   [1,2,3,4,5  (does not terminate)
   [3,20,9]
   7
   [1,2,3,4]
   ```

2. A Haskell function `multall` multiplies all the elements of a given list, for example: `multall [1..4] = 24`.

   Write THREE versions of the function `multall` using different syntactical constructs provided by Haskell: a conditional, guarded equations and finally pattern matching.

   Answer:
   Conditional:

   ```
   multall x = if null x then 1 else head x*multall (tail x)
   ```

   Using guarded equations:

   ```
   multall x
     | null x = 1
     | otherwise = head x * multall (tail x)
   ```

   Using pattern matching:

   ```
   multall [] = 1
   multall (h:t) = h*multall t
   ```

3. Using the higher-order function `foldr`, write an expression that will multiply all the numbers from 1 to 100. Give the type of all the functions used in the expression.

   Answer:

   ```
   foldr (*) 1 [1..100]
   foldr :: (a -> b -> b) -> b -> [a] -> b
   (*) :: Integer -> Integer -> Integer
   ```

4. Write a function `multacc` which is the accumulating parameter version of the following Haskell function:

   ```
   mult x y = if x==0 then 0 else (mult (x-1) y)+y.
   ```

   Give the type of your function and show which arguments would be passed to `multacc` to multiply 5 and 4?

   Answer:

   ```
   multacc x y a = if x==0 then a else multacc (x-1) y (acc+y)
   multacc :: Int -> Int -> Int
   mutlacc x y 1
   ```

5. Using list comprehension, write a Haskell expression that will generate Pythagorean triples (triple of numbers that satisfy Pythagoras' theorem: $a^2 + b^2 = c^2$) for numbers up to 100.

   Answer: Application of ideas taught on list comprehensions.
   A number of slight variants possible (for example $(3, 4, 5)$ and $(4, 3, 5)$ may be considered as the same triple). Full marks will be awarded in either case for a correct solution.

   ```
   [(x,y,z) | x <-[1..100], y <-[1..100],
              z <-[1..100], x^2+y^2 == z^2]
   ```

6. In Haskell, we can write pairs as (t, u), where t and u are any terms. How could we write tuples in Haskell if this feature was not supported by language? (Hint: you might like to think of a solution in the λ-calculus first).

   Answer: Example:

   ```
   mpair x y z = z x y
   mfst x  = x(\a -> \b -> a)
   msnd x  = x(\a -> \b -> b)

   *Main> mfst (mpair 2 4)
   2
   *Main> msnd (mpair 2 4)
   4
   ```

7. Using `map` and a one-off function (written using Haskell's lambda notation), write a function that will swap all pairs of a list of pairs of numbers. Thus, write a function `f` such that, for example, `f [(1,2),(3,4)] = [(2,1),(4,3)]`.

   Answer:

   ```
   map (\(x,y) -> (y,x))
   ```

8. What are the types of the following Haskell expressions:

(a) `(&&) True`

(b) `\f -> \x -> f (f x)`

(c) `hd [True]`

(d) `error`

(e) `\(x,y) -> x&&y`

Answer:

(a) `(&&) True :: Bool -> Bool`

(b) `\f -> \x -> f (f x) :: (t -> t) -> t -> t`

(c) `hd [True] :: Bool`

(d) `error :: [Char] -> a`

(e) `\(x,y) -> x&&y :: (Bool, Bool) -> Bool`