CMPT 383 Comparative Programming Languages
Quiz 1 Solution

This quiz has 20 questions in total, each weighing 1 point. Quiz time: 40 minutes

Student name: _____

1. Which of the following CANNOT be made a functor directly? C
   A. [ ]
   B. Maybe
   C. Either
   D. Data.Map.Map Int

2. Consider the following function

```
foo :: [Int] -> Int
foo xs = fooAux xs 0
  where
    fooAux :: [Int] -> Int -> Int
    fooAux [] y = 1 + y
    fooAux (x:xs) y = 1 + fooAux xs (y + 1)
```

   Is this function tail recursive? B
   A. Yes, it is tail recursive
   B. No, it is not tail recursive

3. Which of the following is NOT correct? B
   A. Given f1 :: a -> b -> a, providing exactly one actual parameter to f1 is allowed
   B. Given f2 :: a -> Float, then f2 cannot take a function of type Int -> Int as input
   C. (Int -> Int) -> Int -> Int is the same as (Int -> Int) -> (Int -> Int)
   D. Given f3 :: (Int -> Int) -> Int, then f3 is a higher-order function

4. Which of the following about IO is NOT correct? C
   A. IO is a functor
   B. IO is an applicative
   C. IO is a monoid
   D. IO is a monad

5. Which of the following is NOT correct? A
   A. main must be of type IO ()
   B. getLine is an IO action
   C. An IO action can be obtained by composing two IO actions
   D. The side effect of an IO action is only carried out when the IO action is performed

6. Which of the following is NOT correct? D
    A. f . g . h $ x = f . g $ h x
    B. f . (g . h) = (f . g) . h
    C. (f . g) x = f $ g x
    D. f (g x) = f . g x

7. Which of the following is NOT correct? A
    A. Suppose a module has declaration `data T = D1 Int | D2 Int` but does not export data constructors D1 and D2, then there is no way to create values of type T in another module
    B. A module named Foo.Bar must be defined in file Bar.hs under the Foo directory
    C. A type declared using the newtype keyword cannot have more than one data constructor
    D. If there is a declaration `type T1 = T2`, then T1 and T2 are interchangeable

8. Consider the declaration `data Pair a = Pair a a` for pairs. If Pair is a functor, what is the type of the corresponding fmap function?

    (a -> b) -> Pair a -> Pair b

9. Consider the declaration `data Pair a = Pair a a` for pairs. If Pair is an applicative, what is the type of the corresponding (<*>) function?

    Pair (a -> b) -> Pair a -> Pair b

10. Consider the declaration `data Pair a = Pair a a` for pairs. If Pair is a monad, what is the type of the corresponding (>>=) function?

    Pair a -> (a -> Pair b) -> Pair b

11. Convert the following function to point-free style. You don't need to write the type signature.

```
bar :: [Int] -> Int
bar xs = sum (filter even (map (max 10) xs))

bar = sum . filter even . map (max 10)
```

12. Use foldr to re-implement the following function. You don't need to write the type signature.

```
baz :: [a] -> [a]
baz xs = foldl (\acc x -> x : acc) [] xs

baz xs = foldr (\x acc -> acc ++ [x]) [] xs
```

13. What types of random values can be obtained using the random function from System.Random?

    All types that are instances of the Random type class

14. Write a lambda that is equivalent to ($ 2)

    ```
    \f -> f 2
    ```

15. Use map and filter to re-implement the following function. You don't need to write the type signature.
    ```
    foo f p xs = [ f x | x <- xs, p x ]
    ```

    ```
    foo f p xs = map f (filter p xs)
    ```

16. Consider the following expression
    ```
    f <$> Just 'a' <*> Just 'b' <*> Just 'c' <*> Just 'd'
    ```
    Suppose the expression type checks and its result is `Just ('b', 'd')`, what could be the implementation of f? You don't need to write the type signature.

    ```
    f _ x _ y = (x, y)
    ```

17. Convert the following function to an equivalent form that does NOT use do notations
    ```
    main = do x <- getLine
              y <- getLine
              putStrLn (x ++ y)
    ```

    ```
    main = getLine >>= (x -> getLine >>= (y -> putStrLn (x ++ y)))
    ```

18. To make [ ] an instance of the Foldable type class, people can implement either one of two functions. What are those two functions?

    foldr and foldMap

19. Consider the declaration `data Pair a = Pair a a` for pairs. What can we do to enable equality comparison between two Pairs only based on the second field (e.g., Pair 1 2 == Pair 2 2 returns True)? Briefly describe your approach in English.

    Make Pair a an instance of Eq and implement the (==) function to compare the 2nd field

20. A type may not be declared as an instance of a particular type class more than once in Haskell. If we really need two instances, what would be the workaround approach?

    Define a "wrapper" of the type using newtype