CS_221 Functional Programming I

(Attempt 2 questions out of 3)

Preliminary remark: By a *function* we mean a *Haskell function*, and by the *definition of a function* we mean the *defining equations* of that function preceeded by its *signature*.

Question 1.

(a) (i) Define a function that sorts a list of integers in ascending order.

(ii) Use a type constraint to define a polymorphic generalization of your sorting function that works for all types that are members of the type class `Ord`.

[8 marks]

(b) Consider the following Haskell type of binary trees labeled by integers:

```
data Tree = Null | Node Tree Int Tree
```

(i) Define a function that computes the sum of the labels of a tree.

(ii) Consider the following function that computes the list of labels of a tree in prefix-order:

```
preord :: Tree -> [Int]
preord Null         = []
preord (Node s n t) = [n] ++ (preord s ++ preord t)
```

Suppose s and t are trees, both different from `Null` and having different labels at their roots.

How many computation steps does it take Haskell to evaluate the expression

`preord` s `==` `preord` t

(assuming that equality of integers is computed in one step)?

Does the number of computation steps depend on the sizes of the trees?

Which evaluation strategy does Haskell apply? Discuss how the computation time would change if a different evaluation strategy were used.

In your calculation use the definition of `(++)` given in question 2 (a) and the following definitions of equality on lists:

```
(==) :: Eq a => [a] -> Bool
[]     == []     =  True
(x:xs) == (y:ys) =  if x == y then xs == ys else False
_      == _      =  False
```

[11 marks]

(c) What are the values of the following expressions?

(i) `(\f -> \x -> f (f x)) (2^) 3`

(ii) `map even [0,1,2,3,4]`

(iii) `[ toLower c | c <- "4 You!" , isAlpha c ]`

[6 marks]

Question 2.

(a) Consider the following definitions:

```
(++) :: [a] -> [a] -> [a]
[]     ++ ys      = ys
(x:xs) ++ ys      = x : (xs ++ ys)

reverse :: [a] -> [a]
reverse []       = []
reverse (x : xs) = reverse xs ++ [x]
```

Prove that

```
reverse (xs ++ ys) = reverse ys ++ reverse xs
```

holds for all finite lists `xs` and `ys`.

You may use without proof that `(++)` is associative and that `xs ++ [] = xs` holds.

[7 marks]

(b) We model a shopping basket as a list of pairs (*name*,*price*) where *name* is a string and *price* is an integer.

Define a function that, for any shopping basket and any integer *maxprice*, computes the list of all names in the basket that are more expensive than *maxprice*.

[6 marks]

(c) Suppose it is your task to design a polymorphic abstract data type of sets of elements of a given type `a`. As possible representations of sets you consider boolean functions and lists:

```
newtype Set1 a = MkSet1 (a -> Bool)
newtype Set2 a = MkSet2 [a]
```

Define, for each of these representations, the following functions, with the appropriate class constraint on `a`, or say why a definition is not possible:

(i) `isEmpty`, testing whether a set is empty;

(ii) `member`, testing whether an element is a member of a set;

(iii) `insert`, inserting an element into a set;

(iv) `complement`, computing the complement of a set `s`, that is, the set of those elements that are not in `s`.

[12 marks]

Question 3.

(a) (i) In Haskell, a type can be defined using one of the keywords, `data`, `type`, or `newtype`. Explain the characteristics of these different ways of defining types. In which cases can functions be defined by pattern matching? You may use simple examples to underpin your explanations.

(ii) What is a Haskell type class? What is it used for? How can a type become an instance of a type class? Give an example.

[8 marks]

(b) (i) Define a polymorphic higher-order function `testAll` that for a given boolean function `p :: a -> Bool` and a list `xs :: [a]` tests whether `p x` evaluates to `True` for all elements `x` of `xs`.

(ii) Use the function `testAll` to define a function `noZeros` that tests whether all elements of a given list of integers are different from 0.

[8 marks]

(c) (i) Briefly explain the meaning of the polymorphic data type `IO a`.

(ii) Write a program

```
passWd ::  IO ()
```

that performs the following action:

- The user is asked to enter a password (write appropriate message on the standard output);
- a string (the password) is read from the standard input;
- the user is asked to retype the password;
- if the two passwords are different, then the action terminates with a message that the password has been rejected,
- otherwise the password is written into the file `password.txt` and a message is written that the password is accepted.

[9 marks]