EXAMINATION:

FUNCTIONAL PROGRAMMING SAMPLE PAPER

The structure of the final paper will be similar in structure, number of questions, choice, level.

Instructions to candidates:
Answer any three questions. All questions carry equal Marks.

Question 1

Part 1.

Select ONE answer for each section.

a) The expression [(1,True),(0,False)] has type:

i) String
ii) [(Int,Bool)]
iii) (Int,Bool)
iv) [Int,Bool]
v) [(Int,Bool),(Int,Bool)] (3 marks)

b) A function of type Int -> Int -> Int:

i) Is a curried function
ii) Takes three arguments
iii) Is a polymorphic function
iv) Is an overloaded function
v) Takes a function as its argument (3 marks)

c) Evaluating sum [x*2 | x <- [1..10], even x] gives:

i) 0
ii) 30
iii) 60
iv) 110
v) 220 (3 marks)

a) Which of the following expressions contains a type error:

i) 1 : [2,3,4]
ii) [] ++ [1,2,3,4]
iii) [1,2,3] ++ 4
iv) [[1,2]] ++ [[3,4]]
v) 1 : 2 : 3 : 4 : [] (3 Marks)

b) Evaluating takeWhile even [2,4,6,7,8] gives:

i) []
ii) [2]
iii) [2,4]
iv) [2,4,6]
v) [2,4,6,8] (3 marks)

c) The function twice defined by twice f x = f (f x) has type:

i) a -> a -> a
ii) (a -> a) -> a -> a
iii) a -> (a -> a) -> a
iv) a -> a -> (a -> a)
v) a -> a -> a -> a (3 marks)

d) The expression Branch (Tip 1) (Tip 2) is a value of the datatype:

i) data Tree = Tip Int | Branch
ii) data Tree = Tip Int | Branch Int Int
iii) data Tree = Tip Tree | Branch Int Int
iv) data Tree = Tip Tree | Branch Tree Tree
v) data Tree = Tip Int | Branch Tree Tree (3 marks)

e) Which of the following equations is true for all finite lists:

f) reverse (map f xs) = map f (reverse xs)
ii) map f (map g xs) = map g (map f xs)
iii) reverse (reverse xs) = reverse xs

iv) map f (map f xs) = map f xs

v) reverse xs = xs (4 marks)

Part 2:

Redefine the functions *map f* and *filter p* using *foldr*

(9 marks)

(Total marks = 34)

Question 2:

a) Consider the following datatype of trees:

data Tree = Leaf Int | Node Tree Tree

i. Write down three different values of type *Tree* with the property that all the leaves in each example contain the integer zero. (3 marks)

ii. Define a function *size* :: *Tree* → *Int* that calculates the number of leaves that are contained in a given tree. (5 marks)

iii. Define a function *depth* :: *Tree* → *Int* that calculates the depth of a tree, where the depth is given by the number of nodes in the longest path from the root of the tree to a leaf in the tree. (6 marks)

iv. Define a function *grow* :: *Int* -> *Tree* that produces a full tree of a given depth in which every leaf contains the integer zero. (3 Marks)

v.

b) The sum of the integers between 1 and 100 can be expressed using the list comprehension notation as sum [x | x <- [1..100]]. Express each of the following using a list comprehension:

Sum of the squares of the integers between 1 and 100

Product of the even integers between 1 and 100

List of all pairs of integers between 1 and 100

Sum of the products of all pairs of integers between 1 and 100

(8 Marks)

c) Define the following functions using *recursion*, stating the type of each of the four functions that you define:

inc – add one to every number in a list of integers

fac – return the factorial of a natural number

evens – return the even numbers in a list of integers

(9 Marks)

(Total marks = 34)

Question 3:

a) Define the class *Monad* of monadic types in Haskell, and explain how this definition can be understood in English. (3)

b) Show how to make the *Maybe* and list types into instances of this class, stating the type of each function that you define. (7)

c) Given the type declaration

data Expr = Val Int | Div Expr Expr

define an evaluation function *eval* :: *Expr* –*Maybe Int* using explicit case analysis (rather than any form of monadic programmming). (7)

d) Show how your definition for *eval* can be rewritten using the *return* and >>= functions of the *Maybe* monad, and explain your definition. (7)

e) Explain how the **do** notation abbreviates a common pattern of monadic programming, and illustrate this pattern by redefining *eval*. (7)

f) Why are the monadic definitions for *eval* preferable? (3)

(Total marks = 34)

Question 4:

a) Define appropriate versions of the library functions:

```
repeat :: a -> [a]
repeat x = xs where xs = x:xs

take :: Int -> [a] -> [a]
take  0 _      = []
take  _ []     = []
take n (x:xs)  = x : take (n - 1) xs

map : (a-> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = f x : map xs
```

for the following type of binary trees:

```
data Tree a = Leaf | Node a (Tree a )  (Tree a)
                deriving Show
```

(3 x 7 Marks)

b) Describe the following terms briefly with examples if this helps:

Pure programming
Functional programming
Lambda calculus
DSLs
Etc.

(13 Marks)
(Total marks = 34)