University of Leicester

CO2008
All candidates

Midsummer Examinations 2015

DO NOT OPEN THE QUESTION PAPER UNTIL INSTRUCTED TO DO SO BY THE CHIEF INVIGILATOR

Department	**Computer Science**
Module Code	**CO2008**
Module Title	**Functional Programming**
Exam Duration	**Two hours**

CHECK YOU HAVE THE CORRECT QUESTION PAPER

Number of Pages	**4**
Number of Questions	**3**
Instructions to Candidates	**Answer all questions.** **All marks gained will be counted.** **Calculators are not allowed.**

For this exam you are allowed to use the following	
Calculators	**Not permitted**
Books/Statutes	**Not permitted**
Additional Stationery	**Not required**

1. (a) What are the most general types of the following terms:

   i. `[(2,0),(1,5)]`

   ii. `(",",',',[']'])`

   iii. `"20+15"`

   iv. `filter`

   v. `[tail ["a"]]` [10 marks]

   (b) Give the (most general) type declarations for the following function definitions:

   i. `zip (a:as) (b:bs) = (a,b):zip as bs`

   ii. `zap (a:as) (b:bs) = (b,a):zap bs as`

   iii. `zop (a:as) (b:bs) = (b,a):zop as bs` [6 marks]

   (c) Evaluate:

   i. `(head [1,2]):[]`

   ii. `1 + head(fst ([1,2],[3,2]))`

   iii. `head [head [1..3] : tail [3..6]]`

   iv. `filter (==3) (tail [3, fst(1,2), 1+2, snd(1,2)])` [8 marks]

   (d) i. What is the most general type of `filter`?

   ii. Write down the function `filter` using recursion.

   iii. Write down the function `filter` using list comprehension. [8 marks]

   (e) i. Write down a function `flatten :: [[a]] -> [a]` that, given a list of lists, flattenenates all its elements into a single list.
   For example: `flatten [[1,4],[3,2]] = [1,4,3,2]`

   ii. Write down a function `elem :: Eq a => a ->[a] -> Bool` such that `elem x list` evaluates to `True` if `x` occurs in `list` and evaluates to `False` otherwise.
   For example: `elem 4 [1,4,3,2] = True` and `elem 5 [1,4,3,2] = False`
   [8 marks]

2. We wish to write Haskell code that can typeset a given text string as a (single) page of lines of at most 30 characters. The space between consecutive words should be a single blank character and the page should be aligned on the left-hand side. The right-hand sided can be ragged.

   To do so we will think of a line abstractly as a list of words, and a page as a list of lines. Hence we introduce the following types:

```
type RawText = String
type Word = String
type Line = [Word]
type Page = [Line]
```

   To typeset a raw text we convert it into a page. First we will split the text into a list of words, then we will group this list of words into a page of lines of appropriate length.

   (a) Write down a function `split :: RawText -> [Word]` that splits a given string into a lists of words. [12 marks]

   To keep things simple, we assume that the raw text is a string of letters and blanks only, possibly with an irregular spacing.

   For example: `split " one two  three   ff " = ["one","two","three","ff"]`

   (b) Write down a function `wrap :: [Word] -> Page` that takes a list of words and groups them from left to right in lists of lines, that, when printed, consist of no more than 30 characters each. [15 marks]

   To keep things simple, we assume there are no words of length greater than 30.

   For example: the term `wrap (split " We want to write Haskell code that can typeset a given text string as a (single) page of lines of at most 30 characters.  ")`
   should result in the page

```
[["We","want","to","write","Haskell","code"],
["that","can","typeset","a","given","text"],
["string","as","a","(single)","page","of"],
["lines","of","at","most","30"],
["characters."]]
```

   To print such a page we will define an auxiliary function `page2string` so that we can typeset that page with the expression `putStr(page2string page)`.

   (c) Write down a function `page2string :: Page -> String` that flattens the words of a page into one single string in which consecutive words from a line are separated by a single space in the string and consecutive lines are separated by a newline character. [5 marks]

   For example: `page2string [["one","two","three","four"],["five","six"]] = "one two three four\nfive six"`.

(d) Finally write down a function `typeset :: String -> IO()` that, given a string, prints it with the desired pretty printing. [3 marks]

For example: `typeset "  We   want to write Haskell code that can typeset a given text string as a (single) page of lines of at most 30 characters.  "`
should give the following output:

```
We want to write Haskell code
that can typeset a given text
string as a (single) page of
lines of at most 30
characters.
```

3. We consider the following datatypes:

```
data Tree a = L a | N a (Tree a) (Tree a) deriving Show
data Dir = F | M deriving Show
type Path = [Dir]
```

As application we think of family trees: for every person in a tree there is a path that describes the relationship between that person and the person at the root of the tree.

For example: in the tree
`N "Anna" (N "Fer-Jan" (L "Willem") (L "Nettie")) (L "Paula")`
the path `[F]` points to the father `Fer-Jan` of `Anna`, and the path `[F,M]` points to the paternal grandmother `Nettie` of `Anna`.

(a) Write down a function `addPath :: Tree a -> Tree (Path,a)` that, given a tree, replaces the label `aa` of each of its node at path `p` by the pair `(p,aa)`. [10 marks]

For example:
`addPath N "Anna" (N "Fer-Jan" (L "Willem") (L "Nettie")) (L "Paula")`
should evaluate to

```
N ([],"Anna") (N ([F],"Fer-Jan") (L ([F,F],"Willem"))
(L ([F,M],"Nettie"))) (L ([M],"Paula")).
```

(b) Write down a function `makesortedlist :: Tree String -> [(Path,String)]` that given a tree labeled by strings produces a list of pairs of type `(Path,String)` in ascending lexicographic order by string. [15 marks]

For example `makesortedlist (N "Anna" (N "Fer-Jan" (L "Willem") (L "Nettie") (L "Paula"))`
should evaluate to the list

```
[([],"Anna"),([F],"Fer-Jan"),([F,M],"Nettie"),([M],"Paula"),
([F,F],"Willem")].
```

END OF PAPER