The University of Nottingham

SCHOOL OF COMPUTER SCIENCE
AND INFORMATION TECHNOLOGY

A LEVEL 1 MODULE, AUTUMN SEMESTER 2006-2007

FUNCTIONAL PROGRAMMING

Time allowed TWO hours

Candidates must NOT start writing their answers until told to do so

Answer QUESTION ONE and THREE other questions

Marks available for sections of questions are shown in brackets in the right-hand margin.

No calculators are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a dictionary to translate between that language and English provided that neither language is the subject of this examination.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn examination paper over until instructed to do so

ADDITIONAL MATERIAL: Haskell Standard Prelude

Question 1 (Compulsory)

1. What are the values of the following expressions (10)
   (a) `[(x,x+2) | x <- [1 .. 4], x > 2]`
   (b) `map k [1,2,5,7] where k x = (x, x ‘mod‘ 3)`
   (c) `phi 4 where phi x = 1 + sum [phi y | y <- [1 .. (x-1)]]`

2. A small document package models a book as a list of lines, a line as a list of words and a word as a string. Thus we make the type definitions

```
type Word = String
type Line = [Word]
type Book = [Line]
```

   Using list comprehensions.

   (a) Define a function `words :: Book -> [Word]` which returns the list of words appearing in a book. You need not remove multiple copies of the same word from your answer. (3)
   (b) Define a function `freq :: Word -> Book -> Int` which returns how often a word appears in a book. (3)
   (c) Define a function `bookFreq :: Book -> [(Word,Int)]` which takes as input a book and returns a list consisting of every word in the book and the number of lines it occurs on. (4)
   (d) Define a function `index :: Word -> Book -> [Int]` which takes a word and a book and returns the line numbers the word appears on. For example, (5)

```
index ’’This’’ [[’’This’’, ’’Town’’],
                 [’’This’’, ’’Cat’’, ’’This’’]
               ]
= [1,2]
```

Question 2:

1. Give an implementation of the sorting algorithm quicksort whose type is `qsort :: Ord a => [a] -> [a]` (5)

2. Give an implementation of the higher order version of the sorting algorithm quicksort, `qsortBy :: Ord b => (a -> b) -> [a] -> [a]` (6)

3. A sporting team is represented by its name and the number of points they have scored in recent games. For example `("Newcastle", [3,3,3,0])` represents a team called `"Newcastle"` which scored 3 points in their last game, 3 points in the previous two games and 0 points in the game before that. This data is modelled by the type definitions

   ```
   type TName = String
   type Points = [Int]
   type Team = (TName,Points)
   ```

   Define the following functions

   (a) `sortPoints :: [Team] -> [Team]` which places one team before another if it has a higher total number of points. (3)

   (b) `sortPointsPlayed :: [Team] -> [Team]` which places one team before another if it has a higher total number of points. If two teams have the same total number of points, then the team which has played fewer games should come first. (5)

   (c) `sortPlayedLast :: Int -> [Team] -> [Team]` which places one team before another if it has a higher total number of points in the last n games, where n is the first input of the function. (6)

Question 3:

A bank stores details on its customers via their national insurance number, their age, and their balance. This gives the following type definitions.

```
type NI = Int
type Age = Int
type Balance = Int
type Person = (NI, Age, Balance)
type Bank = [Person]
```

1. Define a function `retired :: Person -> Bool` which decides if the person is at least 65 years old. (2)
2. Define a function `deposit :: Person -> Int -> Person` which adds the second input to the first input's balance. (4)
3. Define a function `credit :: Bank -> [Person]` which returns those people who are not overdrawn. (3)
4. Define a function `equityAge :: Bank -> (Int,Int) -> Int` which returns the total deposits held at the bank by people between the two ages specified in the second input of the function. (4)
5. In society there are lots of banks and the population can be represented by their national insurance numbers. Hence we define

   ```
   type Market = [Bank]
   type Pop = [NI]
   ```

   Define a function `creditNI :: NI -> Market -> Int` which returns the total deposits held by the person whose national insurance number is the first input at all the banks in the second input. (5)
6. Define a function `bankFree :: Pop -> Market -> Pop` which returns those people who do not have any bank accounts. (7)

Question 4:

1. Define the polymorphic type `Tree a` of binary trees which store data of type `a` at only the leaves of the tree. (4)

2. Define expressions `tree1` and `tree2` which represent the following trees.(4)

```
      .            .
     / \          / \
    8   3        .   8
                / \
               4   6
```

3. Define functions

```
lowest, highest ::  Ord b => Tree b -> b
```

   which take a binary tree as input and return the least and greatest elements stored in the tree. For example `lowest tree2 = 4` and `highest tree2 = 8`. (4)

4. Using the functions `lowest` and `highest`, define a function

```
ordered ::  Ord b => Tree b -> Bool
```

   which returns true if every element in a left subtree is no more than any element in a right subtree. Trees which are leaves are ordered. For example `ordered tree1 = False` while `ordered tree2 = True`. (6)

5. Paths are used to locate data stored within a tree. A path is a list of directions from the root of the tree to a piece of data stored in the tree. Each direction is either `L` signifying the left-hand branch or `R` signifying the right hand branch. Thus we make the type definitions

```
data Dir  = L | R
type Path = [Dir]
```

   Define a function `paths ::  Tree a -> [Path]` which calculates the paths in a tree. For example (7)

```
paths tree1 = [[L], [R]]
paths tree2 = [[L,L], [L,R], [R]]
```

Question 5:

Write clear and precise descriptions of each of the following concepts that arise within functional programming. Make sure your answer explains the practical benefits of these concepts and also includes both simple and more complex examples.

1. Recursion (8)
2. List Comprehensions (7)
3. Algebraic types (10)

You may not use examples taken from other questions in this paper, your answers to them, or the standard prelude.