INFORMATION 1 - FUNCTIONAL PROGRAMMING

Wednesday 8 December 2010

09:30 to 11:30

Convener: J Bradfield
External Examiner: A Preece

INSTRUCTIONS TO CANDIDATES

1. Note that ALL QUESTIONS ARE COMPULSORY.

2. DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS. Take note of this in allocating time to questions.

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY
1. (a) Write a function \( f :: [\text{Int}] -> \text{Int} \) to find the product of one half of each even number in a list. For example,

\[
\begin{align*}
f [1,2,3,4,5,6] &= 6 \\
f [2,4,6,8] &= 24 \\
f [4,-4,4] &= -8 \\
f [2,2,2] &= 1 \\
f [1,3,5] &= 1
\end{align*}
\]

Your definition may use basic functions, list comprehension, and library functions, but not recursion. Credit may be given for indicating how you have tested your function. \([12 \text{ marks}]\)

(b) Write a second function \( g :: [\text{Int}] -> \text{Int} \) that behaves like \( f \), this time using basic functions and recursion, but not list comprehension or other library functions. Credit may be given for indicating how you have tested your function. \([12 \text{ marks}]\)

(c) Write a third function \( h :: [\text{Int}] -> \text{Int} \) that also behaves like \( f \), this time using one or more of the following higher-order library functions:

\[
\begin{align*}
\text{map} &:: (a -> b) -> [a] -> [b] \\
\text{filter} &:: (a -> \text{Bool}) -> [a] -> [a] \\
\text{foldr} &:: (a -> b -> b) -> b -> [a] -> b
\end{align*}
\]

You may also use basic functions, but not list comprehension, other library functions, or recursion. Credit may be given for indicating how you have tested your function. \([12 \text{ marks}]\)
2. (a) Write a polymorphic function \( p :: [a] \rightarrow [a] \) that swaps adjacent elements in a list of even length. The behaviour of the function is unspecified if given a list of odd length. For example:

\[
\begin{align*}
    p \ "abcdef" &= "badcfe" \\
    p \ [1,2,3,4] &= [2,1,4,3] \\
    p \ [0,0,0,0] &= [0,0,0,0] \\
    p \ "" &= ""
\end{align*}
\]

Your function may use basic functions, list comprehension, and library functions, but not recursion. Credit may be given for indicating how you have tested your function. [16 marks ]

(b) Write a second function \( q :: [a] \rightarrow [a] \) that behaves like \( p \), this time using basic functions and recursion, but not list comprehension or library functions. Credit may be given for indicating how you have tested your function. [16 marks ]
3. (a) A scalar is a single integer, and a vector is a pair of integers.

```haskell
type Scalar = Int
type Vector = (Int, Int)
```

Write functions

```haskell
add :: Vector -> Vector -> Vector
mul :: Scalar -> Vector -> Vector
```

that add two vectors by adding corresponding components of the vectors, and multiply a scalar and a vector by multiplying each component of the vector by the scalar. For example,

```haskell
add (1,2) (3,4) == (4,6)
mul 2 (3,4) == (6,8)
```

[4 marks]

(b) The following data type represents terms that compute vectors. A term is a vector consisting of two scalars, the sum of two terms, or the multiplication of a scalar by a term.

```haskell
data Term = Vec Scalar Scalar
    | Add Term Term
    | Mul Scalar Term
```

Write a function `eva :: Term -> Vector` that takes a term and computes the corresponding vector. For example,

```haskell
eva (Vec 1 2) == (1,2)
eva (Add (Vec 1 2) (Vec 3 4)) == (4,6)
eva (Mul 2 (Vec 3 4)) == (6,8)
eva (Mul 2 (Add (Vec 1 2) (Vec 3 4))) == (8,12)
eva (Add (Mul 2 (Vec 1 2)) (Mul 2 (Vec 3 4))) == (8,12)
```

Credit may be given for indicating how you have tested your function.

[14 marks]

(c) Write a function `sho :: Term -> String` that converts a term to a string. Vectors should be printed as a pair of integers in parentheses, sums and products should be written infix surrounded by parentheses. For example,

```haskell
sho (Vec 1 2) == "(1,2)"
sho (Add (Vec 1 2) (Vec 3 4)) == "((1,2)+(3,4))"
sho (Mul 2 (Vec 3 4)) == "(2*(3,4))"
sho (Mul 2 (Add (Vec 1 2) (Vec 3 4))) == "((2*((1,2)+(3,4))))"
sho (Add (Mul 2 (Vec 1 2)) (Mul 2 (Vec 3 4)))
    == "((2*(1,2))+(2*(3,4)))"
```

Credit may be given for indicating how you have tested your function.

[14 marks]