Department of Information and Computing Sciences, Faculty of Science, UU.
Made available in electronic form by the TBC of A–Eskwadraat
In 2010/2011, the course INFOFP was given by Doaitse Swierstra.

Functioneel Programmeren (INFOFP)
8 november 2011

Question 1: The function *foldl'* *(1 point)*

Give the type and the definition of the function *foldl'*. Give an example where its use is profitable and an example where its use is not giving the desired effect at all.

Question 2: Fair enumeration *(1 point)*

Define a value *enumInts* :: [(*Int*,*Int*)] in which the distance from an occurrence of any value from the set $\{(x, y) \mid x \in Int, y \in Int\}$ to the beginning of the list is a finite value (note that *Int*'s can also be negative).

Question 3: Permutations *(1 point)*

Write a function *permutations* :: $[a] \to [[a]]$ which returns all permutations of its parameter.

Question 4: Side effects *(1 point)*

Someone writes the following program and does not get any output.

```
import System.Random
createRandomValues  = sequence (repeat randomIO)
printRandomValues n = do randomValues <- createRandomValues
                         print (take n randomValues)
main = printRandomValues 10
```

Rewrite the program such that it does what the code suggests, i.e. printing 10 random numbers.

Question 5: Is *tja* correct? *(1 point)*

Remark of the TBC: The original code used in this question was wrong. The following code is the corrected code.
Given the data type

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

we define the function *tja*:

```
tja t = let tja' (Leaf a)   n ls = (0, if n==0 then a:ls else ls)
            tja' (Node l r) n ls = let (lm,ll) = tja' l (n-1) rl
                                       (rm,rl) = tja' r (n-1) ls
                                   in ((lm `min` rm) + 1, ll)
            (m, r) = tja' t m []
        in r
```

If this code computes something explain what it computes (small example?); if it does not compute anything explain why this is the case.

Question 6: The function *enumBf* *(1 point)*

Write a function *enumBf* :: *Tree a* → [*a*] which returns a list which contains the *a*-values from the leaves resulting from a breadth-first enumeration (i.e. leaves at a lower depth occur earlier in the list). Hint: use a helper function *enumBf'* :: [*Tree a*] → [*a*].

Question 7: Parsing *(2 points)*

We can define a somewhat simplistic data type *XML* and a parser for it:

```
type Tag = String
data XML = Tag Tag [XML]
         | Content String
```

pXML = (*pOpenTag* >>= (λ*t* → *Tag t* <$> *pMany pXML* <∗ *pCloseTag t*))
<|> *Content* <$> *pString*
Write the functions *pOpenTag* and *pCloseTag*. Write a parser *pXML'* which also recognises attributes, and returns the result as a value of type *XML'*. You may assume that *pString* takes care of escaping special characters. Assume also that *pString* and *pSym* remove any trailing whitespace (i.e. you do not have to worry about spaces, newlines, tabs, etc).

```
data XML'   = Tag' Tag Attrs [XML']
            | Content' String
type Attrs' = [(String, String)]
```

An example input might be:

```
<TABLE COLS="1" BORDER="0" CELLSPACING="4" CELLPADDING="5">
  <TR>
    <TD COLSPAN="2" WIDTH="100%" BGCOLOR="#99CCFF" ALIGN="CENTER">
      <B>Functional programming in the Netherlands</B>
    </TD>
  </TR>
</TABLE>
```

Question 8: Heaps *(2 points)*

A *heap* is a data structure described by a data type quite similar to a search tree:

```
data Heap a = Top a (Heap a) (Heap a)
            | Empty
```

with the property that the *a* value in a *Top* node dominates (⩾) all the values contained in its two children, which have this property themselves too.

1. Write a function *checkHeap* :: *Ord a* ⇒ *Heap a* → *Bool* which returns *True* if its argument is a *heap*, and *False* otherwise. Hint: you may want to write a helper function *checkHeap'* :: *Ord a* ⇒ *a* → *Heap a* → *Bool*.

2. Write a function *mergeHeaps* :: *Ord a* ⇒ *Heap a* → *Heap a* → *Heap a* which combines its two arguments into a *heap*.

3. Write the function *enumHeap* :: *Ord a* ⇒ *Heap a* → [*a*] such that the value *r*:

   v = *enumHeap* . *foldr mergeHeaps Empty* $ [*Top x Empty Empty* | *x* ← [1..10]]

   evaluates to [10, 9, 8, 7, 6, 5, 4, 3, 2, 1], i.e. the elements stored in the heap come out in reversed sorted order.