EXAM FUNCTIONAL PROGRAMMING
Tuesday the 5th of November 2013, 9.00 h. - 12.00 h.

Name:	
Student number:	

Before you begin: Do not forget to write down your name and student number above. If necessary, explain your answers (in English or Dutch). For multiple choice questions, clearly circle what you think is the best answer. Use the empty boxes under the other questions to write your answer and explanations in. Use the empty paper provided with this exam only as scratch paper (kladpapier). At the end of the exam, only hand in the filled-in exam paper. Answers will not only be judged for correctness, but also for clarity and conciseness. A total of one hundred points can be obtained; divide by 10 to obtain your grade. Good luck!

1. We want to implement a function that computes the size of a value of a given type by means of type classes. Hence, we introduce:

```
class Sizable t where
  size :: t → Int
  -- An Int is an atomic value:
instance Sizable Int where
  size i = 1
```

   (i) Give an instance declaration for sizing lists. The size of a list is obtained by adding the sum of the sizes of its elements to the number of elements in the list.

   .../**8**

2. In this question we deal with a function $perms :: [\,a\,] \to [[\,a\,]]$ which returns all the permutations (i.e., all possible orderings) of the argument list.

   (i) What are the permutations of [1,2,3,4] that start with 1?

   $\boxed{\ldots/\mathbf{4}}$

   (ii) Explain how you can compute $perms\ (x : xs)$ from $perms\ xs$ (for example by using concrete values for x and xs)

   $\boxed{\ldots/\mathbf{6}}$

   (iii) Now, write the function $perms :: [\,a\,] \to [[\,a\,]]$

   $\boxed{\ldots/\mathbf{9}}$

3. In this question we again see the *perms* function of the previous question. Even if you failed to come up with an implementation for *perms*, you may still be able to answer this one.

   (i) Give a QuickCheck property that can check that all lists in the outcome of *perms* have the right length.

   .../**8**

   (ii) Give the same property as at (i), but now so that it will only check this for input lists that have no duplicate elements.

   .../**6**

   (iii) Chances may be small that a randomly generated list of *Int*s has no duplicates, and QuickCheck may give up in despair. Define a generator to generate random lists with no duplicate elements.

   .../**5**

4. $\boxed{\ldots/\mathbf{20}}$ The following multiple choice questions are each worth 5 points.

(i) Someone tries to write a function $\mathit{revDigits} :: \mathit{Int} \to \mathit{Int}$ that "reverses the digits in an *Int*"; so 123 is mapped onto 321. Which is the correct solution?

a. $\mathit{revDigits}\ i = \mathit{foldl}\ (\lambda \mathit{ds}\ x \to x : \mathit{ds})$ `""` $(\mathit{show}\ i)$

b. $\mathit{revDigits}\ i = \mathit{foldr}\ (\lambda x\ \mathit{ds} \to \mathit{ds} \mathbin{+\!\!+} [x])$ `""` $(\mathit{show}\ i)$

c. $\mathit{revDigits}\ i = \mathit{revDigits}'\ i\ 0$
  where $\mathit{revDigits}'\ 0\ r = r$
    $\mathit{revDigits}'\ i\ r = \mathit{revDigits}'\ (i\ \text{`}\mathit{div}\text{`}\ 10)\ (r * 10 + i\ \text{`}\mathit{mod}\text{`}\ 10)$

d. $\mathit{revDigits}\ i = \mathit{revDigits}'\ i\ 0$
  where $\mathit{revDigits}'\ 0\ r = r$
    $\mathit{revDigits}'\ i\ r = \mathit{revDigits}'\ (i\ \text{`}\mathit{mod}\text{`}\ 10)\ (r * 10 + i\ \text{`}\mathit{div}\text{`}\ 10)$

(ii) What is the type of *concat* . *concat*?

a. $[[[a]]] \to [a]$

b. $[a] \to [[a]] \to [a]$

c. $[[a]] \to [[a]] \to [[a]]$

d. none of the above

(iii) Given are the following two statements:

I The side effects possible because of IO make Haskell an impure language

II Using *seq* can never make your program slower

a. Both I and II are true

b. Only I is true

c. Only II is true

d. Both I and II are false

(iv) Given the definition of strict function application \$! from the slides, consider the following two statements:

I snd \$! $(\bot_1, \bot_2)$ equals $\bot_2$

II length \$! $\mathit{map}\ \bot\ [1,2]$ equals $\bot$

a. Both I and II are true

b. Only I is true

c. Only II is true

d. Both I and II are false

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

$$\begin{aligned}&\textbf{data}\ \mathit{Heap}\ a = \mathit{Top}\ a\ (\mathit{Heap}\ a)\ (\mathit{Heap}\ a)\\ &\qquad\qquad\qquad\mid\ \mathit{Leaf}\end{aligned}$$

with the so called *heap property* that the a value in a *Top* node is larger than or equal to the values in the roots of its child *Heap*s, which have this property themselves too. An example of a heap is

$$\begin{aligned}\mathit{aHeap} = \mathit{Top}\ 10\ &(\mathit{Top}\ 7\ \mathit{Leaf}\\ &\qquad(\mathit{Top}\ 4\ \mathit{Leaf}\ \mathit{Leaf}))\\ &(\mathit{Top}\ 3\ (\mathit{Top}\ 2\ \mathit{Leaf}\ \mathit{Leaf})\\ &\qquad\mathit{Leaf})\end{aligned}$$

(i) Write a function $\mathit{checkHeap} :: \mathit{Ord}\ a \Rightarrow \mathit{Heap}\ a \rightarrow \mathit{Bool}$ which returns *True* if its argument has the required propery, and *False* otherwise. **Hint**: you may want to write a helper function $\mathit{checkHeap}' :: \mathit{Ord}\ a \Rightarrow a \rightarrow \mathit{Heap}\ a \rightarrow \mathit{Bool}$.

$\boxed{\ldots/\mathbf{8}}$

(ii) Write a function $\mathit{getFromHeap} :: \mathit{Ord}\ a \Rightarrow \mathit{Heap}\ a \rightarrow \mathit{Maybe}\ (a, \mathit{Heap}\ a)$, which – provided the heap is non-empty– returns the largest a value from its *Heap a* argument, together with a *Heap a* containing the rest of the values of its *Heap a* argument, and *Nothing* if the *Heap a* argument is a *Leaf*. Make sure the resulting *Heap a* again satisfies the heap property! **Hint**: the new root of the heap is either the root of the left subtree or the root of the right subtree.

$\boxed{\ldots/\mathbf{8}}$

6. Given the following definitions of *reverse*, and $(+\!\!+)$:

$$\begin{array}{lll} reverse :: [a] & \rightarrow & [a] \\ reverse \ \ [\,] & = & [\,] \\ reverse \ \ (x : xs) & = & reverse\ xs +\!\!+ [x] \end{array}$$

$$\begin{array}{lll} (+\!\!+) :: [a] \rightarrow [a] \rightarrow [a] \\ [\,] \quad +\!\!+ ys & = & ys \\ (x : xs) +\!\!+ ys & = & x : (xs +\!\!+ ys), \end{array}$$

(i) Prove by induction that $xs = xs +\!\!+ [\,]$. Do not forget to explain every step in your proof!

$\boxed{\ldots/\mathbf{5}}$

(ii) Prove by induction that $reverse\ (xs +\!\!+ ys) = reverse\ ys +\!\!+ reverse\ xs$. You may use the result of part (i), and a Lemma that says that $(xs +\!\!+ ys) +\!\!+ zs = xs +\!\!+ (ys +\!\!+ zs)$ in your proof. Again, do not forget to explain every step in your proof!

$\boxed{\ldots/\mathbf{13}}$