## EXAM FUNCTIONAL PROGRAMMING

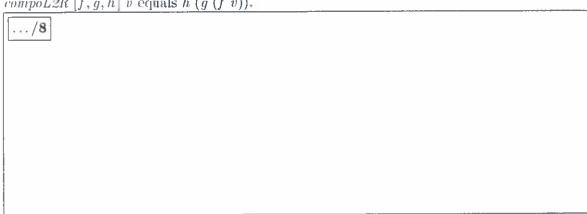
Tuesday t	he 30th of September 2014, 11.00 h 13.00 h.
Name: Student	number:
explain your street is the cone and explain the empty particular than did concist Good luck In any functions/	begin: Do not forget to write down your name and student number above. If necessary, our answers (in English or Dutch). For multiple choice questions, clearly circle what you think and only) best answer. Use the empty boxes under the other questions to write your answer nations in; if you run out of space, you can use the empty sixth page of the exam. Use the per provided with this exam only as scratch paper (kladpapier). At the end of the exam, only ne filled-in exam paper. Answers will not only be judged for correctness, but also for clarity seness. A total of one hundred points can be obtained; divide by 10 to obtain your grade. It of your answers below you may (but do not have to) use the following well-known Haskell operators: id, concat, foldr (and variants), map, filter, const, flip, fst, snd, not, (.), element, take While, drop While, head, tail, (++), lookup and all members of the type classes Eq. (5), Show and Read.
1. (i)	Write a function intersperse :: $a \to [a] \to [a]$ , which places its first argument between the elements of its second argument; i.e. intersperse 'a' "xyz" should return "xayaz". You must use direct recursion.  [/8]
(ii)	Give an alternative definition of intersperse without direct recursion, using higher-order functions. /8

2. The operator (.) composes two functions. We want to generalise this and implement a function that composes a list of functions compoR2L of type  $[(a \rightarrow a)] \rightarrow a \rightarrow a$ . For example, the call compoR2L[f,g,h] v computes the value f(g(hv)).

(i) Implement compoR2L using foldr.

implement composition and John .
/8

(ii) Implement compoL2R, that composes functions in the opposite direction. In other words, compoL2R [f, g, h] v equals h (g (f v)).



(iii) What do you get when you evaluate compoL2R [not, even] 3?

```
.../5
```

3. Given is the following definition of a so-called Trie a

```
data Trie\ a = Leaf\ a\ |\ Branch\ a\ [(Char, Trie\ a)]
```

The idea of a *Trie* is that *every branch and leaf* contains a payload value of type a, and that the children of a branch are indexed by a value of type *Char*. An example of a *Trie Int* is the following:

```
ex = Branch 40 [('a', Branch 20 [('a', Leaf 1),
('b', Leaf 2)]),
('b', Branch 30 [('a', Leaf 3),
('c', Leaf 4)])
```

1

(i)	Write a function $sumIntTrie$ :: $Trie\ Int \rightarrow Int$ that adds all the payloads (of type $Int$ ) together. For example, $sumIntTrie\ (Leaf\ 3)$ , should return 3, and $sumIntTrie\ ex$ should return 100.		
	/10		
(ii)	Write a function $searchTrie :: [Char] \to Trie \ a \to Maybe \ a$ , that follows a path down the tree as indicated by the first argument, and just returns the payload of the $branch$ or $leaf$ it reaches in this way, and $Nothing$ otherwise. For example, $searchTrie$ [] $ex$ gives $Just\ 40$ , $searchTrie$ ['b', 'a', 'b'] $ex$ gives $Just\ 30$ , $searchTrie$ ['b', 'a'] $ex$ returns $Just\ 3$ , and $searchTrie$ ['b', 'a', 'h'] $ex$ returns $Nothing$ . Here you may use a function $clookup::Char \to [(Char,b)] \to Maybe\ b$ such that $clookup\ c$ $ps$ returns $v$ if $(c,v)$ is the first pair in $ps$ in which $c$ is the first component, and $Nothing$ if no such pair exists.		
	/10		
(iii)	A problem with the definition of <i>Trie</i> is that the type system does not forbid values like Branch 45 [('a', Leaf 33), ('a', Leaf 78)]. Give a definition for <i>Trie</i> a that does not have that problem.		
	/5		

- - (i) Let f be any function of type  $Int \to Int$ . Which expression has the same value as the following list comprehension?

$$[f \ x \mid x \leftarrow [1, 6], even \ x]$$

- a. map f (filter even [1..6])
- b. filter even (map f [1..6])
- c. f (map even [1...6])
- d. filter f (map even [1...6])
- (ii) Given is the following unnecessarily complicated function definition:

$$f g = f 0 g$$
  
where  $f g h \mid g < length h = foldr (const (+1)) 0 (h !! g) + f (g + 1) h$   
 $\mid otherwise = 0$   
 $const f g = f$ 

Which of the following implementations is equivalent?

- a. sum . map length
- b. foldr (+) 0 . map (const 1)
- c. foldr ((+) length) 0
- d. foldl1 (+) . map length
- (iii) I Both function application and the → in function types associate to the left so that Currying becomes possible.
  - II Function application has precedence over all operators.
  - 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) What is the type of map . foldr?

a. 
$$(a \to a \to a) \to [a] \to [[a] \to a]$$

b. 
$$(a \rightarrow a \rightarrow a) \rightarrow [b] \rightarrow [b \rightarrow a]$$

c. 
$$(b \to a \to a) \to [b] \to [[a] \to a]$$

d. 
$$(b \to a \to a) \to [a] \to [b] \to a$$

5.	(i)	Explain why the expression $\lambda x \to (x \ ['1'], x '1')$ is type incorrect.
		/5
		Determine the type of map filter. You should not just write down the type below, but also explain how you arrived at that type (for example, in the way that this is done in the lecture notes of this course).
		/15

