Last Name:				
First Name:				
Matriculatio	n Number:			

Exercise	Points	Score
Types	12	
Evaluation	11	
Programming	15	
I/O and Modules	7	
Σ	45	

- You have 90 minutes time to solve the exercises.
- The exam consists of 4 exercises, for a total of 45 points (so there is 1 point per 2 minutes).
- The available points per exercise are written in the margin.
- Don't remove the staple (Heftklammer) from the exam.
- Don't write your solution in red color.

Remarks:

- This exam was designed as a closed book exam, i.e., no notes, slides, books, computers, . . . were allowed.
- Blank paper for making notes have been made available to all participants.
- This is an old exam, where all exercises can be solved with your current knowledge, except for exercise 2(c).

Exercise 1: Types Consider the following Haskell code:	12
data Type a = Empty Node a Int (Type a) deriving Eq	
<pre>c = Node d = \ x -> Node x x Empty f x y z = if x == Empty then y else z g x = if x > Empty then "Hello" else replicate 10 '!'</pre>	
In each multiple choice question, exactly one statement is correct. Marking the correct statement is worth 3 points, giving no answer counts 1 point, and marking multiple or the wrong statement results in 0 points.	
(a) The most general type of c is:	(3)
\square Type a -> a -> Int -> Type a -> Type a	
\square a -> Int -> Type a -> Type a	
\square Eq a => a -> Int -> Type a -> Type a	
\square Eq a => a -> Int -> Type a	
\Box c is not type-correct.	
(b) The most general type of d is:	(3)
□ a -> Type a	
☐ Eq a => a -> Type a	
□ a -> Type (a,a)	
☐ Int → Type Int	
□ d is not type-correct.	(9)
(c) The most general type of f is ☐ Eq a => Type a -> b -> b	(3)
☐ Type a -> b -> b	
\Box (Eq a, Eq b) => Type a -> b -> b	
\Box Eq a => Type a -> a -> a	
\Box f is not type-correct.	
(d) The most general type of g is	(3)
☐ Type String -> String	(-)
☐ Ord a => Type a -> String	
☐ Eq a => Type a -> String	
\square Type a -> String	
\square g is not type-correct.	

	cise 2: Evaluation ider the following Haskell code:	
rop rop rop rop rop	<pre>last_A, drop_last_B, drop_last_C, drop_last_D, drop_last_E :: [a] -> [a] last_A xs = take (length xs - 1) xs last_B = drop 1 . reverse last_C = reverse . tail . reverse last_D xs = map fst (zip xs (tail xs)) last_E xs = [xs !! j i <- [1 length xs], let j = i - 1]</pre>	
1	Assume the input is a non-empty finite list $[x_1, \ldots, x_n]$. Then most of the drop_last_X-functions return the list $[x_1, \ldots, x_{n-1}]$. Write down all drop_last_X-functions that return a different list and also give the result of these functions.	
	Next we consider the empty list as input. Write down the result of drop_last_X [] for X = B,C,E and provide a step by step evaluation of drop_last_D []. As a reminder, here are the definitions of zip and tail. tail (_ : xs)	
	21p (X . AS) (y . yS) - (X,y) . 21p AS yS	
ĺ	Now assume the input is an infinite list. Write down all drop_last_X-functions which satisfy that drop_last_X [0] evaluates to [0]. Note: this part cannot be solved with the current knowledge, i.e., lecture 10, WS 2021/22.	

Exercise	3:	Programn	nine
	•		

15

(2)

(3)

(3)

Consider a function find which given a key k and a list of key-value pairs, returns v if (k, v) is the first entry in the list with key k, or nothing if no such pair exists.

Examples:

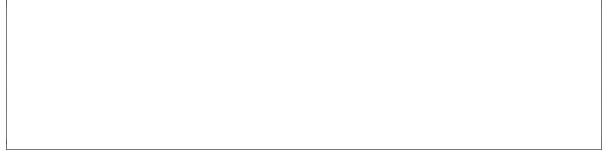
- find 5 [(3, "a"), (5, "b"), (5, "c"), (2, "g")] = Just "b"
- find 'c' [('a',1), ('z',26)] = Nothing
- (a) Give a suitable type-definition of find. In particular, the examples above should be type-correct, and one should be able to implement find with your type.

i			
ļ.			
ł			
L			

(b) Provide a recursive definition of find that does not use any library functions on lists, except for the list constructors.

(c) Provide a non-recursive definition of find that is based on list-comprehensions. (3)

(d) Provide a non-recursive definition of find that is based on foldr.



VO Functional Programming Test Exam December 13, 2021 (e) Write a function bad_item :: [(String,String)] -> Maybe String which returns an item that is rated poorly, if such an item exists. • The input list of rated items is always given in pairs of the form (item, rating), e.g., as in [("coffee", "medium"), ("lemonade", "poor"), ("tea", "good"), ...]. • If there are many poorly rated items, return the one which is last in alphabetical order. You may assume that all item names are provided in lower-case letters. • In the definition you may use find from above and standard list functions like sort, map, reverse, ..., but neither list-comprehensions nor filter.

(4)

7

Exercise 4: I/O and Modules

Consider the following Haskell module.

module Area where

```
area :: Double -> Double
area r = pi * r * r
```

Write a Haskell program (outside of the module Area) which asks the user for a radius and then prints the area of the circle with that radius, *precisely* as formatted in the two lines between the prompt>...-lines.

prompt> ./my_program # start program
Enter radius: 6.72
Area of circle with radius 6.72 is 141.8692976878693.
prompt> # program has ended

- The program should be compilable via ghc --make.
- The user made exactly one input, namely the first occurrence of the number 6.72.
- For the calculation, the method area has to be invoked.

I			
I			
	1		