WS 20/21 – 1. Klausur (01.02.2021)

◉ Korrekte Single-Choice Antworten

1: Evaluation of Expressions (Single Choice)

Consider the following program:

```
data Foo = ConsA Int Bool | ConsB String

h x Nothing = undefined
h (ConsA x b) (Just y) = if b then x else y

inf x = inf (x + 1)
```

and the following expressions.

```
t0 := h (ConsA (inf 5) False) (if False then Nothing else Just (inf 1))

t1 := h (ConsA (inf 5) False) (Just (inf (1 + 1)))

t2 := h (ConsA (inf 5) False) (Just (inf 1))

t3 := h (ConsA (inf ((5 + 1) + 1) False) (Just (inf 1))

t4 := if False then inf 5 else inf 1

t5 := h (ConsA (inf (5 + 1)) False) (Just (inf 1))

t6 := h (ConsA (inf 5) False) (if False then Nothing else Just (inf 2)

t7 := h (ConsA (inf 5) False) (if False then Nothing else Just (inf (1 + 1))

t8 := h (ConsA (inf 6) False) (Just (inf 1))
```

How does the evaluation of t0 start?

Hint: Remind yourself of the evaluation order for pattern matching.

[✓ a. t0 = t2 = t4]

b. t0 = t2 = t1

c. t0 = t5 = t3

d. t0 = t5 = t8

e. t0 = t7 = t6

2: Analyzing Programs (Single Choice)

Consider the following program:

```
1.  module Faulty(SomeType) where

2.  import qualified Data.Char as D -- Data.Char contains toUpper

3.  toUpperString (x : xs) = D.toUpper x : toUpperString xs

4.  two = length [\ x -> undefined, \ x y -> 5]

5.  toUpper c = succ (succ c)

6.  toUpperReverseString xs = map toUpper (reverse Xs)
```

Identify those parts of the program that cause compile errors.

○ a. Only line 6.

○ b. Lines 5 and 3.

○ c. Lines 6 and 4.

[✓ d. Lines 1 and 6.]

○ e. Only line 1.

3: Higher Order Functions (Single Choice)

Consider the following function:

```
foo :: [a] -> [b] -> [(a,b)]
foo xs ys = fst $ foldr helper ([], ys) xs where
  helper x (zs, y:ys) = ((x,y) : zs, ys)
```

The expression `foo xs ys` is equivalent to:

○ a. `zip ys xs`

○ b. `zip (reverse xs) ys`

○ c. `zip xs ys`

[✓ d. The expression is not equivalent to any expression that is given in the other answers.]

○ e. `zip xs (reverse ys)`

4: Programming with Lists and Type Classes

Task 1 (8 points)

Define a function

```
printList :: Show a => [a] -> String
```

such that

```
printList [] = "()"

printList [1,2,3] = "(1 and 2 and 3)"

printList "hallo" = "('h' and 'a' and 'l' and 'l' and 'o')"
```

Note that also the blanks must be inserted correctly.

Task 2 (8 points)

Consider a datatype declaration

```
data NewList a = CreateList [a]
```

Make NewList an instance of the Show-class such that show (CreateList xs) is the string show xs, except that the first and the last character have been removed.

Examples:

```
show (CreateList [1,2,3]) = "1,2,3" -- and not "[1,2,3]"

show (CreateList "hallo") = "hallo" -- and not "\"hallo\""
```

Task 3 (4 points)

Is it possible to make an instance declaration of lists of type [a] such that the following expression evaluates to True?

```
show [1,2,3] == "(1 and 2 and 3)"
```

Provide your answer in the program by defining a constant answerTask3 :: Bool

Solution

Task 1

```
printList :: Show a => [a] -> String
printList  [] = "()"
printList  (x:xs) = "(" ++ show x ++ middle ++ ")"
   where middle = concat $ map (\ x -> " - " ++ show x) xs
```

Task 2

```
data NewList a = CreateList [a]

instance Show a => Show (NewList a) where
    show (CreateList xs) = removeFirstLast (show xs)

removeFirstLast :: [a] -> [a]
removeFirstLast xs = tail (reverse (tail (reverse xs)))
```

Task 3

```
answerTask3 :: Bool
answerTask3 = False
```

Die Deklarierung einer bereits existierenden Klasse (in diesem Fall show) ist nicht möglich und führt zu einer Fehlermeldung.

5: Programming with Characters and Higher Order Functions

Task 1 (9 points)

Define a function

```
rotLetter :: Char -> Char
```

to rotate a letter. To be more precise, rotating a letter is the operation that maps every lower-case letter (a letter in `"a..z"`) to the next letter in the alphabet, except that `’z’` is mapped to `’a’`. Rotating a character that is not a lower-case letter returns the same character.

Example: `map rotLetter "Hello Homer!" = "Hfmmp Hpnfs!"`

Hint: In this task it is not required to write 27 different cases.

Task 2 (6 points)

Define a function

```
nTimes :: (a -> a) -> Int -> (a -> a)
```

such that `nTimes f n x` applies `f` `n`-times to `x`.

Example: `nTimes ((+) 2) 5 7 = 17` and `nTimes ((:) x) n [] = replicate n x`

You may assume that the second argument of `nTimes` is always non-negative.

Task 3 (5 points)

Define a function

```
encrypt :: Int -> String -> String
```

such that in `encrypt n xs` every letter of `xs` is rotated `n`-times via `rotLetter`.

Example: `encrypt 3 "Hello Homer!" = "Hhoor Hrphu!"`

It must hold that if `n` is between 0 and 26 then `encrypt (26 - n) (encrypt n xs) = xs`.

Solution

Task 1

```
rotLetter :: Char -> Char
rotLetter x
  | x >= 'a' && x < 'z' = succ x
  | x == 'z' = 'a'
  | otherwise = x
```

Task 2

```
nTimes :: (a -> a) -> Int -> (a -> a)
nTimes  f 0 = (\ x -> x)
nTimes  f n = f . nTimes  f (n - 1)
```

Task 3

```
encrypt :: Int -> String -> String
encrypt n xs = map (nTimes rotLetter n) xs
```

6: Programming with List Comprehensions

Task 1 (7 points)

Define a function `minList :: Ord b => (a -> b) -> [a] -> a` such that `minList f xs` is any element `x` of `xs` such that `f x <= f y` for all `y` in `xs`. Here, you can assume that `xs` is non-empty.

For example: `minList negate [3,7,5] = 7`.

saas

Task 2 (9 points)

Consider the following type definition for representing items in a store

```
type Item a = (a, Integer)   -- (item identifier, weight)
```

where in a list of items, each item identifier is unique and all weights are positive.

Define a function

```
selections :: Ord a => [Item a] -> [((a, a, a), Integer)]
```

such that the resulting list consists of precisely those entries `((i1, i2, i3), w)` that satisfy the following criteria:

- `i1`, `i2`, `i3` are item identifiers that occur in the input list,
- the item identifiers are sorted: `i1 < i2 < i3`,
- the number `w` is the total weight of the three items, and
- `w` is at least 131.

Example:

```
selections [
   ("Oranges", 26),
   ("Bananas", 54),
   ("Iron", 16),
   ("Paper", 60)
 ]
 = [(("Bananas", "Oranges", "Paper"), 140)]
```

Task 3 (4 points)

Combine `selections` and `minList` to write a function

```
bestCombination :: Ord a => [Item a] -> (a, a, a)
```

that computes some combination of exactly three items such that the total weight is at least 131 and such that the total weight is minimal among all such combinations. You can assume that the input list contains at least three different items whose total weight is at least 131.

Example:

```
bestCombination [
   ("Ding", 45),
   ("Foo", 40),
   ("Bar", 54),
   ("Word", 60)
 ]
 = ("Bar", "Ding", "Foo")
```

Solution

Task 1

```haskell
minList :: Ord b => (a -> b) -> [a] -> a
minList  f [x] = x
minList  f (x : xs) =
   let m = minList  f xs
    in if f x < f m then x else m
```

Task 2

```haskell
type Item2 a = (a, Integer)

selections :: Ord a => [Item2 a] -> [((a, a, a), Integer)]
selections  xs = [((i1, i2, i3), w) |
       (i1, w1) <- xs,
       (i2, w2) <- xs,
       (i3, w3) <- xs,
       i1 > i2 && i2 > i3,
    let w = w1 + w2 + w3, w >=131]
```

Task 3

```haskell
bestCombination :: Ord a => [Item a] -> (a, a, a)
bestCombination xs = let
   fst $ minList snd (selections  xs)
```

7: Programming with Input and Output

Task 1 (4 points)

Define a function `isPalindrome :: Eq a => [a] -> Bool` that checks whether a list is a palindrome, i.e., reading it from left-to-right is the same as reading it from right-to-left. For example, `"HANNAH"` is a palindrome, whereas `"JONAS"` and `[1,2,3,4]` are not.

Task 2 (11 points)

Define a Haskell program that contains a function `main :: IO ()`. When executed the program should read one line after the other (via `getLine`). It should stop its execution on input `"quit"`, and then outputs the number of palindromes that have been entered.

Here is an example dialog where all lines except for the last have been entered by the user.

```
hello

there are not that many palindromes, but I know a few:

able was i ere i saw elba

mada m I m adam

neve r o dd o r even

That's it from my side

quit

I found 3 palindrome(s)!
```

Make sure that in your program, the output is precisely formatted as indicated in the last line.

Solution

Task 1

```haskell
isPalindrome :: Eq a => [a] -> Bool
isPalindrome xs = xs == reverse xs
```

Task 2

```haskell
main :: IO ()
main = main2 0

main2 :: Int -> IO ()
main2 x = do
    s <- getLine
    if s == "quit" then
        putStr ("I found " ++ show x ++ " palindrome(s).")
    else let n = if isPalindrome s then 1 else 0
        in main2 (x+n)
```