Institute of Computer Science, Faculty of Mathematics and Computer Science, UU.
Made available in electronic form by the TBC of A−Eskwadraat
In 2004/2005, the course FP was given by Prof. dr. S.D. Swierstra.

Solutions[1] Functioneel Programmeren (FP) 14 april 2005

The exam consists of 4 multiple choice questions (1 point each) and 2 open questions (3 points each). A wrong multiple choice answer will give a negative result ($-\frac{1}{4}$ point), whereas omitting the answer results in 0 points. Therefore, guessing is not recommended.

Exercise 1

Which of the following items is true for the following definition:

```
class Eq a where
  (==), (/=) :: a → a → a → Bool
  x /= y = not (x == y)
  x == y = not (x =/ y)
```

a) In a class definition it is not allowed to define functions in terms of each other.

b) This is exactly the definition of the class `Eq` from the Haskell report.

c) Because `Eq` is built-in into Haskell it can also be used to compare functions

d) The function definitions are not allowed here, since they belong to the **instance** declarations and not the class declaration.

Correct answer: **b**.

Exercise 2

Using GHCi the Haskell expression 2 + True results in the error message:

```
No instance for (Num Bool)
  arising from use of '+' at <interactive>:1:1
Probable fix: add an instance declaration for (Num Bool)
```

If we follow the hint of the system we have amongst others to:

a) Define a function `fromInteger` that maps `True` to some integer value.

b) Define a function `(+)` with type `Integer → Bool → Int`.

c) Define a function for `fromInteger` that has the type `Integer → Bool`.

d) Both b and c.

Correct answer: **c**.

[1]These solutions were made with great precaution. In case of errors, the TBC cannot be held responsible. However, she will be glad to be informed: tbc@A-Eskwadraat.nl

Exercise 3

In the Haskell Prelude the list constructor [] has been made an instance of the class `Monad`:

```
instance Monad [] where
  ma >>= a2mb = concat (map a2mb ma)
  return a    = [a]
```

Which of the following equals `[ f x y | x ← expr1, y ← expr2 ]`?

a) `do return (f x y) where do x ← expr1 y ← expr2`

b) `do x ← expr1 y ← expr2 f x y`

c) `do x ← expr1 y ← expr2 return (f x y)`

d) `do y ← expr2 x ← expr1 return (f x y)`

Correct answer: **c**.

Exercise 4

Which of the following is true?

a) If we want to `show` a value of type `[a]` we always have to make sure that `show` is also defined for values of type `a`.

b) We can call `show` on values of type `[a]`, without having defined `show` for `a`, as long as `a` itself is also a list type.

c) If we define `show` for `[a]`, then `show` for values of type `a` is automatically constructed.

d) We cannot define `show` for the polymorphic type `[a]` since we cannot make this work for all possible types `a` at the same time.

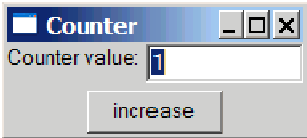
Correct answer: **a**.

Exercise 5

Write a function `compositions :: [Int] → Int → Int` that computes in how many different ways we can use the stamp (Dutch: postzegel) values from the first argument to build the value given as second argument (assume we have an infinite supply of all denominations).

A correct answer could be:

```
compositions _ 0 = 1
compositions [] _ = 0
compositions (x:xs) n | n < 0 = 0
                      | n > 0 = compositions (x:xs) (n - x)
                      +
                      compositions xs n
-- or assuming stamps /= 0
compositions stamps n | n < 0 = 0
                      | n == 0 = 1
                      | n > 0 = sum [ compositions stamps (n - x) | x ← stamps ]
```



Exercise 6

Write a Haskell program that creates a window containing a button and a text field. The text field contains a number that is increased when the button is pressed. Make sure your layout is the same as the layout in the screenshot.

A correct answer could be:

```
main = start $
do { counterV ← varCreate 1
   ; f ← frame [ text := "Counter" ]
   ; l ← entry f [ text := "1" ]
   ; b ← button f [text := "increase", on command := increase counterV l ]
   ; set f [ layout := column 5 [ row 5 [ label "Counter value:", widget l ]
                                , hfloatCenter $ widget b
                                ]
           ]
   }
where increase counterV counterLabel =
  do { i ← get counterV value
     ; set counterV [ value := i + 1 ]
     ; set counterLabel [ text := show i ]
     }
```