Departement Informatica en Informatiekunde, Faculteit Bètawetenschappen, UU.
In elektronische vorm beschikbaar gemaakt door de TBC van A–Eskwadraat.
Het college INFOFP werd in 2008/2009 gegeven door Doaitse Swierstra.

# Functioneel Programmeren (INFOFP)
# 4 maart 2009

Het tentamen bestaat uit 6 multiple-choice vragen (1 punt elk) en 2 open vragen (2 punten elk).

## Opgave 1

Wat is het type van `map concat`

a) `[[[a]]] -> [a]`

b) `[[[a]]] -> [[a]]`

c) `[[a]] -> [a]`

d) geen van a) t/m c)

## Opgave 2

Welke van de volgende uitspraken is waar m.b.t. de expressie `takeWhile True (repeat True)`:

a) resultaat is gelijk aan `repeat True`

b) resultaat is gelijk aan `[]`

c) evaluatie gaat in een loop, en er verschijnt geen resultaat

d) is niet goed getypeerd

## Opgave 3

Wat is het correcte resultaat van de expressie `foldr ((-) . (+2)) 2 [2, 2, 2]`?

a) 0

b) 2

c) $-2$

d) $-6$

## Opgave 4

Wat is de correcte uitspraak over het volgende programma?

```
main = do c <- return 'c'
          c <- return (do c <- getChar
                          return (putChar c))
          c
          c
          return ()
```

a) Leest twee karakters in (met echo) en doet verder niets.

b) Geeft als uitvoer `'c'cc`

c) Leest een karakter in en drukt dat eindeloos af

d) Leest twee karakters in (met echo) en drukt die weer af.

## Opgave 5

Iemand vertaalt het volgende programma, zodat het geen lijstcomprehensies meer bevat. Wat is het correcte resultaat?

```
segs xs = [] : [t | i <- inits xs, t <- tails i, not (null t)]
```

a)
```
[] : concat (map f . map g) (inits xs)
   where f i = tails i
         g t = if null t then [] else [t]
```

b)
```
[] : filter (not.null) . concat (map f . map g) (inits xs)
   where f i = tails i
         g t = [t]
```

c)
```
[] : concat (map f (inits xs) )
   where f i = concat (map g (tails i) )
               where g t = if not(null t) then [] else [t]
```

d)
```
[] : concat (map f (inits xs) )
   where f i = concat (map g (tails i) )
               where g t = if not(null t) then [t] else []
```

## Opgave 6

Welk van de volgende types is equivalent aan een binaire zoekboom? Voor de volledigheid:

```
data GTree f a = GLeaf
               | GNode a (f (GTree f a))
data Paar a = Paar a a
```

a) `type ZB a = GTree Paar (a,a)`

b) `type ZB a = GTree [] a`

c) `type ZB a = GTree Paar a`

d) `type ZB a = GTree (a,a) a`

## Opgave 7

Given the type

```
data Tree = Node Tree Int Tree
          | Leaf
```

Write a function `smallPathsOnly :: Int -> Tree -> [[Int]]` that efficiently produces all paths from the root to the leafs of which the sum of the elements on that path is less than the given `Int` parameter.

## Opgave 8

Schrijf een functie `lines :: Int -> [String] -> [String]` die een lijst met woorden opsplitst in een aantal regels, waarbij de regels niet langer mogen zijn dan de meegegeven `Int` waarde. In de resulterende regels zijn de woorden gescheiden door een `' '`. Woorden mogen niet afgebroken worden, en de lengte van het langste woord is hoogstens de meegegeven `Int` waarde.