

LOGISCH EN FUNCTIONEEL PROGRAMMEREN

deeltentamen Functioneel Programmeren

18 december 2002, 9–11 uur, Educatorium zaal theta

- Schrijf op elk ingeleverd blad je **naam**, en op het eerste blad ook je **collegekaartnummer** en het aantal ingeleverde bladen.
 - De zes meerkeuzevragen tellen elk voor 6 punten, de vier open vragen tellen elk voor 16 punten.
 - Donderdagmiddag is er nog practicumbegeleiding voor de tweede practicumopdracht, die uiterlijk vrijdag(nacht) moet worden ingeleverd.
 - Zou je de evaluatie-enquete willen invullen, die je vindt op www.cs.uu.nl/education/enquete ?
 - Na de vakantie gaat het gedeelte Logisch Programmeren verder op *dinsdag*middag 7 januari.
 - De uitslag van tentamen en practicum verschijnt zodra het is nagekeken op de website www.cs.uu.nl/docs/vakken/1fp.
-

1. De functie `foldr` heeft drie parameters. Als je hem partieel parametrizeert met één parameter, houdt je een functie met twee parameters over. Wat is het type van de partiële parametrisatie `foldr (++)` ?
 - A: $[a] \rightarrow [a] \rightarrow [a]$
 - B: $[a] \rightarrow [a] \rightarrow [[a]]$
 - C: $[a] \rightarrow [[a]] \rightarrow [a]$
 - D: dit is een typeringsfout
2. Welke expressie heeft dezelfde waarde als de lijstcomprehensie `[f x | x <- [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])`
3. Welke expressie heeft *niet* dezelfde waarde als `foldr f 0 [1,2,3]` ?
 - A: `1 'f' (2 'f' (3 'f' 0))`
 - B: `f 1 (f 2 (f 3 0))`
 - C: `f (f (f 0 3) 2) 1`
 - D: `(f 1 . f 2 . f 3) 0`
4. Welk type is hetzelfde als het type $(a \rightarrow b) \rightarrow c \rightarrow d$?
 - A: $(a \rightarrow b, c) \rightarrow d$
 - B: $(a \rightarrow b) \rightarrow (c \rightarrow d)$
 - C: $((a \rightarrow b) \rightarrow c) \rightarrow d$
 - D: $a \rightarrow b \rightarrow c \rightarrow d$

5. Bekijk de volgende definities:

```
getallen = iterate ((+)1) 0
klein x = x<100
as = take 100 getallen
bs = takeWhile klein getallen
cs = filter klein getallen
```

Wat is het geval?

- A: `as`, `bs` en `cs` geven alledrie een eindige, afgesloten lijst
- B: `as` en `bs` geven een eindige afgesloten lijst, maar `cs` geeft een nog-niet afgesloten lijst.
- C: `as` geeft een eindige afgesloten lijst, maar `bs` en `cs` geven een nog-niet afgesloten lijst.
- D: `as`, `bs` en `cs` geven alledrie een nog-niet afgesloten lijst.

6. Bekijk de volgende data-definitie:

```
data Boom a = Blad
            | Tak a (Boom a) (Boom a)
```

Wat is het type van de expressie `Tak 1 Blad` ?

- A: `Int -> Boom Int`
- B: `Boom Int`
- C: `Boom Int (Boom Int)`
- D: `Boom Int -> Boom Int`

7. Schrijf een functie `insert` die een getal invoegt in een van klein naar groot geordende lijst, zo dat de lijst na afloop nog steeds geordend is.

Schrijf een functie `sorteer` die een lijst van getallen sorteert, door de elementen met behulp van `insert` één voor één in te voegen, beginnend met een lege lijst. Gebruik expliciete recursie.

Schrijf de functie `sorteer` nog eens, ditmaal echter zonder recursie maar met gebruik van een standaard hogere-ordefunctie uit de prelude.

8. Schrijf een functie

```
splits :: [a] -> [ ([a],[a]) ]
```

die alle mogelijkheden oplevert waarop een lijst in twee delen kan worden opgesplitst.

Hieronder staat een voorbeeld van hoe de functie kan worden aangeroepen, en wat de uitkomst dan moet zijn:

aanroep	uitkomst
<code>splits [1,2,3,4]</code>	<code>[([], [1,2,3,4])</code> <code>, ([1], [2,3,4])</code> <code>, ([1,2], [3,4])</code> <code>, ([1,2,3], [4])</code> <code>, ([1,2,3,4], [])</code> <code>]</code>

Hint: maak de functie recursief, en kijk goed naar het voorbeeld hoe het resultaat van de recursieve aanroep uitgebreid kan worden tot het gewenste resultaat. Definieer eventueel ook nog een hulpfunctie.

9. Bekijk het volgende type:

```
data Tree a = Leaf
            | Fork a [Tree a]
```

Op alle `Fork`-posities staat dus één waarde, en splitst de tree zich niet in twee takken, maar in een hele lijst van takken.

Schrijf nu een functie `overall`, met als parameter een functie en een tree, die de functie toepast op alle waarden die in de tree zijn opgeslagen. Geef ook het type van `overall` aan.

10. Geef de definitie van de constante `tafels` die gelijk is aan het volgende tweezijdig oneindige blok getallen:

```
[ [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ....
  , [ 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, ....
  , [ 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, ....
  , [ 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, ....
  , [ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, ....
  , ....
```

Geef daarna ook een definitie van een functie `diagonaal`, die de diagonaal van zo'n dubbel-oneindig blok getallen geeft. De aanroep van `diagonaal tafels` zou bijvoorbeeld de lijst met kwadraten `[1, 4, 9, 16, 25,]` opleveren.

1 Helium standaardfuncties

Operator-prioriteiten

```
infixr 9  .
infixl 9  !!
infixr 8  ^, ^., **.
infixl 7  *, *., 'quot', 'rem', 'div', 'mod', /., /
infixl 6  +, -, +., -.
infixr 5  ++, :
infix  4  ==, /=, <=, <, >, >=, ==., /=., <=., <., >., >=.
infixr 3  &&
infixr 2  ||
infixr 0  $, $!
```

Functies op Booleans en characters

```

otherwise    :: Bool
not          :: Bool -> Bool
(&&), (||)   :: Bool -> Bool -> Bool
and, or      :: [Bool] -> Bool
any, all     :: (a->Bool) -> [a] -> Bool

isAscii, isControl, isPrint, isSpace           :: Char -> Bool
isUpper, isLower, isAlpha, isDigit, isAlphanum :: Char -> Bool

toUpper, toLower :: Char -> Char
ord            :: Char -> Int
chr           :: Int -> Char

data Maybe a = Nothing | Just a
maybe :: b -> (a -> b) -> Maybe a -> b
data Either a b = Left a | Right b
either :: (a -> c) -> (b -> c) -> Either a b -> c
data Ordering = LT | EQ | GT

```

Numerieke functies

```

(+), (-), (*), (/), (^)           :: Int -> Int -> Int
(<), (<=), (>), (>=), (==), (/=)  :: Int -> Int -> Bool
abs, signum, negate               :: Int -> Int
rem, div, mod, quot                :: Int -> Int -> Int
subtract, gcd, lcm, min, max       :: Int -> Int -> Int
odd, even                          :: Int -> Bool
sum, product, maximum, minimum    :: [Int] -> Int

pi :: Float
(+.), (-.), (*.), (/.), (^.), (**.) :: Float -> Float -> Float
(<.), (<=.), (>.), (>=.), (==.), (/=.) :: Float -> Float -> Bool
exp, log, sin, cos, tan            :: Float -> Float

intToFloat :: Int -> Float
round, truncate, floor, ceiling :: Float -> Int

```

Polymorfe functies

```

undefined :: a
id        :: a -> a
const    :: a -> b -> a
fix      :: (a -> a) -> a
($)      :: (a->b) -> (a->b)
strict  :: (a->b) -> (a->b)

```

```

(.)      :: (b->c)    -> (a->b) -> (a->c)
uncurry  :: (a->b->c) -> ((a,b)->c)
curry    :: ((a,b)->c) -> (a->b->c)
flip     :: (a->b->c) -> (b->a->c)
until    :: (a->Bool) -> (a->a) -> a -> a
until'   :: (a->Bool) -> (a->a) -> a -> [a]
fst      :: (a,b)    -> a
snd      :: (a,b)    -> b

```

Functies op lijsten

```

head     :: [a] -> a
last     :: [a] -> a
tail     :: [a] -> [a]
init     :: [a] -> [a]
(!!)    :: [a] -> Int -> a
index    :: Int -> [a] -> a

take, drop      :: Int      -> [a] -> [a]
splitAt         :: Int      -> [a] -> ([a],[a])
takeWhile, dropWhile :: (a->Bool) -> [a] -> [a]
takeUntil      :: (a->Bool) -> [a] -> [a]
span, break    :: (a->Bool) -> [a] -> ([a],[a])

length        :: [a] -> Int
null          :: [a] -> Bool
elemBy, notElemBy :: (a->a->Bool) a -> [a] -> Bool

(+++)       :: [a] -> [a] -> [a]
iterate     :: (a->a) -> a -> [a]
repeat     :: a      -> [a]
cycle      :: [a]    -> [a]
replicate  :: Int -> a -> [a]
reverse    :: [a]    -> [a]

concat      :: [[a]] -> [a]
map         :: (a->b)  -> [a] -> [b]
concatMap  :: (a -> [b]) -> [a] -> [b]
filter     :: (a->Bool) -> [a] -> [a]

foldl      :: (a->b->a) -> a -> [b] -> a
foldl'     :: (a->b->a) -> a -> [b] -> a
foldr      :: (a->b->b) -> b -> [a] -> b
foldl1     :: (a->a->a) -> [a] -> a
foldr1     :: (a->a->a) -> [a] -> a
scanl      :: (a->b->a) -> a -> [b] -> [a]

```

```

scanl'  :: (a->b->a) -> a -> [b] -> [a]
scanr   :: (a->b->b) -> b -> [a] -> [b]
scanl1  :: (a->a->a) ->      [a] -> [a]
scanr1  :: (a->a->a) ->      [a] -> [a]

zip      :: [a] -> [b]                -> [(a,b)]
zip3     :: [a] -> [b] -> [c]        -> [(a,b,c)]
zipWith  :: (a->b->c)                  -> [a]->[b]->[c]
zipWith3 :: (a->b->c->d)                -> [a]->[b]->[c]->[d]
unzip    :: [(a,b)] -> ([a],[b])
unzip3   :: [(a,b,c)] -> ([a],[b],[c])

```

Functionies op strings

```

type String = [Char]
readInt      :: String -> Int
readUnsigned :: String -> Int
words       :: String -> [String]
lines       :: String -> [String]
unwords     :: [String] -> String
unlines     :: [String] -> String

```

Gelijkheid en ordening

```

eqChar    :: Char -> Char -> Bool
eqBool    :: Bool -> Bool -> Bool
eqString  :: String -> String -> Bool
eqMaybe  :: (a->a->Bool) -> Maybe a -> Maybe a -> Bool
eqList    :: (a->a->Bool) -> [a] -> [a] -> Bool
eqTuple2  :: (a->a->Bool) ->
              (b->b->Bool) -> (a,b) -> (a,b) -> Bool

ordString :: String -> String -> Ordering
ordChar   :: Char -> Char -> Ordering
ordInt    :: Int -> Int -> Ordering
ordList   :: (a->a->Ordering) -> [a] -> [a] -> Ordering

elemBy    :: (a->a->Bool) -> a -> [a] -> Bool
notElemBy :: (a->a->Bool) -> a -> [a] -> Bool
lookupBy  :: (a->a->Bool) -> a -> [(a,b)] -> Maybe b

```

Input/output functies

```

unsafePerformIO :: IO a -> a
return          :: a -> IO a
(>>=)          :: IO a -> (a -> IO b) -> IO b

```

```
sequence    :: [IO a] -> IO [a]
sequence_  :: [IO a] -> IO ()
putChar     :: Char -> IO ()
putStr     :: String -> IO ()
putStrLn   :: String -> IO ()
print      :: (a -> String) -> a -> IO ()
getLine    :: IO String
```