KS i Programmeringsparadigm 2013, del 2: Haskell
2013-11-19 08.15-09.00 med efterföljande kamraträttning

1. (3 p)

   Vad blir resultatet då man kompilerar/kör nedanstående?
   Motivera ditt svar noga annars ger det noll poäng!

   a) (1 p)
   ```
   name n = 37 + n
   name True = 34
   ```

   .....

   .....

   .....

   b) (1 p)
   ```
   whatever 0 = 37
   whatever n = True
   ```

   .....

   .....

   .....

   c) (1 p)
   ```
   meanwhile x = 34
   meanwhile 0 = 35
   ```

   .....

   .....

   .....

Lösning:

a) Kompilerar inte, orsaken är att `True` och `n` är av olika typ.

b) Kompilerar inte, orsaken är att den inte kan returnera olika typer av värden.

c) Kompilerar, men resultatet vid körning är alltid 34 då första fallet även tar 0.

2. (3 p)

   a) (2 p) Förklara vad som menas med sharing.

   .....

   .....

   .....

   .....

   b) (1 p) Vilken är den huvudsakliga anledningen till att sharing används vid lat evaluering?

   .....

   .....

   .....

Lösning:

a) Sharing innebär att ett värde refereras till med en pekare istället för att det kopieras.

b) Att minimera beräkningar. Ett uttryck beräknas högst en gång, trots att det till synes förekommer fler gånger. T.ex. `square tal = tal * tal, square (2*3)`, då beräknas `2*3` bara en gång.

3. (3 p)

   Förklara detaljerat vad som händer vid anropet
   `(\ x y -> x+y) 3`
   dvs där det är möjligt berättar du vilka värden som tilldelas till vad,
   vilka operationer/funktioner som utförs samt vad resultatet blir.

   .....

   .....

   .....

   .....

   .....

   .....

   .....

Lösning:

Värdet 3 knyts till den första parametern som här kallas x, ingen operation/funktion utförs, resultatet är en funktion med en inparameter
dvs $(\backslash y -> 3 + y)$ returneras.

4. (6 p)

   Givet följande kod:

```
minFunktion = (foldr (+) 0. (map ( \ x -> x*10).filter (\x -> odd x)))
```

   a) (1 p) Vad blir resultatet av anropet `minFunktion [1,2,3]`?

   .....

   b) (4 p) Skriv nu om `minFunktion` så att det är en rekursiv funktion och därmed inte använder `foldr`, `map`, `filter` eller .-operatorn.

   .....

   .....

   .....

   .....

   c) (1 p) Vilken är den huvudsakliga fördelen med svansrekursion jämfört med rekursion.

   .....

   .....

Lösning:

a) 40

b)
```
minFunktion2 [] = 0
minFunktion2 (x:xs)
   | odd x = (x*10) + (minFunktion2 xs)
   | otherwise = minFunktion2 xs
```

c) Svansrekursion är mer minneseffektiv (eftersom stacken inte behöver användas)

5. (5 p)

Betrakta följande kod: (bra att veta: `ord :: Char -> Int`)

```
import Data.Char
class MyComp a where
  comp :: a -> a -> Bool
instance MyComp Char where
  comp tal1 tal2 = (ord tal1) <= (ord tal2)
```

Skriv en ny instansfunktion som med hjälp av den givna koden jämför om en sträng är (lexikografiskt) mindre eller lika med en annan sträng. Exempelvis:

```
*Main> comp "abbas" "abba"
False
*Main> comp "abba" "abbas"
True
```

.....

.....

.....

.....

.....

.....

.....

Lösning:

```
instance MyComp a =>
         MyComp [a] where
         comp [] _ = True
         comp _ [] = False
         comp (x:xs) (y:ys)
            | comp x y && comp y x = comp xs ys
            | comp x y = True
            | otherwise = False
```