Kontrollskrivning i DD1361 Programmeringsparadigmer: Haskell

2010-10-26

Vid godkänt på denna KS har du klarat Haskellavsnittet på alla framtida tentor i DD1361. Det krävs 12 poäng för godkänt. Du kan tillgodoräkna dig en bonuspoäng för varje godkänd Prologlab. Inga hjälpmedel utöver skrivmedel är tillåtna.

Namn:
Poäng: +bonus =

1. Uttrycket `map length` ... (1p)

   (a) tar en lista och applicerar `length` på varje element.
   (b) returnerar en lista av funktioner *length*.
   (c) skapar en lista av längden `length`.
   (d) skaper en datastruktur som givet en längd returnerar ett element.

2. I Prelude hittar man funktionen `take :: Int -> [a] -> [a]`.
   Vad är typen för uttrycket `take 100`? (1p)

   (a) `[a]`
   (b) `a -> [a]`
   (c) `[a] -> [a]`
   (d) `Int -> [a]`

3. Vad betyder uttrycket `length . (filter odd)`? (1p)

   (a) En funktion som antingen använder `length` eller `(filter odd)`.
   (b) En funktion som beräknar längden av resultatet av `filter odd`.
   (c) En funktion som beräknar längder av elementen i en lista, och behåller de som är udda.
   (d) Inget, det blir ett kompileringsfel.

4. Uttryck i ord vad följande funktion gör. (1p)

```
fkn :: [a] -> [a] -> [a]
fkn [] _ = []
fkn _ [] = []
fkn (x:xs) (y:ys) = (x : y : fkn xs ys)
```

5. Skriv en funktion `truefalse` som returnerar en oändlig lista av omväxlande sanna och falska värden. (2p)

   Kod:

6. Skriv en funktion `vartannat` som returnerar vartannat element ur en lista. (Det är lätt att det blir fel på sista elementet om man inte är noggrann.) (3p)

   Exempel:

```
Main> vartannat []
[]
Main> vartannat [1,2,3]
[1,3]
Main> vartannat [1,2,3,4]
[1,3]
```

   Kod:

7. Definiera en datastruktur för träd: (4p)

```
data TMap a = Leaf a
            | Children [TMap a]
              deriving Show
```

   Då kan man till exempel skapa testdata så här:

```
t1 = Children [Children [Leaf 1, Leaf 2], Leaf 3]
```

   Skriv nu en funktion `tmap :: (a -> b) -> TMap a -> TMap b` som tillämpar första argumentet på varje löv i indataträdet. Din lösning ska kunna fungera så här på testdatat ovan:

```
ghci> tmap odd t1
Children [Children [Leaf True,Leaf False],Leaf True]
```

   Kod:

8. Skriv en funktion `both` som tar argumenten `f1`, `f2`, och `lis`, där `f1` och `f2` är funktioner med signaturen `f1, f2 :: a -> Bool` och `lis` är en lista av typen `[a]`, så att `both` returnerar de element i `lis` för vilka både `f1` och `f2` är sanna. (2p)

   Kod:

9. I Haskell-listor kan man inte blanda element av olika typer. Antag att vi vill komma undan detta genom att definiera en datatyp för heterogena listor så här:

```
data HElem = IE Int
           | FE Float
           | RE Rational
```

   där IE, FE, och RE markerar heltals-, flyttals-, respektive rationellt-element. Ett exempel på en lista är då

```
tal = [IE 17, FE 1.0, RE 1.2, IE 23]
```

   (a) Skriv en funktion `baraheltal :: [HElem] -> [Int]` som returnerar precis de heltalselement, i rätt ordning, som finns i indatalistan. (2p)

   (b) Skriv en funktion `red_ut` som returnerar en tretupel med heltal, flyttal och rationella tal i varsin lista. (3p)

   Exempel:

```
Main> baraheltal [IE 17, FE 1.0, RE 1.2, IE 23]
[17, 23]
Main> red_ut [IE 17, FE 1.0, RE 1.2, IE 23]
([17,23],[1.0],[6 % 5])
```

   Kod: