Kontrollskrivning i DD1361 Programmeringsparadigmer: Haskell

24 oktober 2011

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 Haskellaboration. Inga hjälpmedel utöver skrivmedel är tillåtna.

Blandade frågor (1 p/ fråga)

1. Vad menas med en ren funktion (pure function)?
Svar: Funktionen har inga sidoeffekter.
2. Vad menas med en polymorf funktion?
Svar: Funktionen har samma definition för alla typer till skillnad mot en överlagrad typ som har samma namn men olika definitioner för olika typer. I Haskell har en polymorf funktion en eller flera typvariabler t.ex. *length*

Evaluering (2 p)

Haskell använder sig av så kallad lat evaluering. Visa hur anropet *andraGradsPolynom(12.0 + 0.1)* evalueras med lat evaluering där:

```
andraGradsPolynom :: (Num t) => t -> t
andraGradsPolynom x = 3 + 5*x + 4*x*x
```

Svar:

```
andraGradsPolynom (12.0 + 0.1)
    =                          Börja med yttre nivån, applicera andraGradsPolynom!
3 + 5*x + 4*x*x
    =                          Pekare från x till minneseutrymme som lagrar (12.0 + 0.1)
                               12.0 + 0.1 beräknas till 12.1.
    =                          3 + 5*x + 4*x*x beräknas för x = 12.1
649.14                         OBS! Man behöver ej svara detta värde men däremot
                               visa evalueringsstegen.
```

Strängar (2 p)

Skriv en funktion för ett ersätta % (kommentartecken i Matlab) med # (kommentartecken i Python) i en textrad (som är av typen sträng). Exempel:

```
haskell> replace "square (3) % Anropar funktionen square med 3"
"square (3) # Anropar funktionen square med 3"
```

Svar:

```
myCheck '%' = '#'
myCheck   x = x
replace row = map myCheck row
```

Rekursion (3 p)

Skriv en funktion *makePolynom* som med rekursion genererar ett generellt polynom och därefter beräknar dess värde. Argumenten till funktionen *makePolynom* är en lista med polynomets koefficienter och ett x-värde.
Exempel: Anropet *makePolynom* [3, 5, 4] 1
konstruerar andragradspolynomet $f(1) = 3 + 5 * 1^1 + 4 * 1^2$ som sedan beräknas till 12.
Svar:

```
makePolynom :: [Double] - > Double -> Double
makePolynom [c] _ = c
makePolynom (c:cs) x = c + x*(makePolynom cs x)
```

Funktion som returnerar en funktion (1 p)

Skriv en funktion *myPolynom* som returnerar ett generellt polynom som konstruerats med funktionen *makePolynom*. Det vill säga *myPolynom* returnerar en funktion. Exempel:
Anropet *myPolynom* [3,5,4] konstruerar andragradspolynomet $f(x) = 3 + 5 * x^1 + 4 * x^2$ (men som inte utför några beräkningar med ett x-värde).

Svar:

```
myPolynom koeff = makePolynom koeff
```

Funktionsvärden med hjälp av listomfattning (2 p/fråga)

1. Skriv en funktion *yValues* där du använder listomfattning för att beräkna funktionsvärden av polynomet *myPolynom(x) [3,5,4]* där $x = x_0, x_0 + h, x_0 + 2 * h, ..., x_n - h, x_n$. Argument till funktionen *yValues* är x_0, x_n och h och typsignaturen är:
$yValues :: Double -> Double -> Double -> [Double]$

Svar:

```
yValues x0 xn h = [(myPolynom [3,5,4]) x | x <- [x0, (x0+h) .. xn]]
```

2. Funktionen *yValues* kan göras mer generell genom att lägga till ett argument så att typsignatuern blir:
$yValues :: (Double -> Double) -> Double -> Double -> Double -> [Double]$
men vad menas med $(Double -> Double)$ i typsignaturen?
Svar:
Det är en funktion som tar en Double och returnerar en Double.

3. Hur ska den mer generella funktionen $yValues$ skrivas? Svar:

```
yValues func x0 xn h = [(func [3,5,4]) x | x <- [x0, (x0+h) .. xn]]
```

Klasser (4 p)

I Haskell kan man använda typklasser t.ex.

```
class Incrementable a where
      inc :: a -> a
instance Incrementable Integer where
      inc x = x + 1
instance Incrementable Char where
      inc c = chr ((ord c) + 1)
```

1.(1 p) I koden ser vi att funktionsnamnet *inc* används till två funktioner! Vad kallas detta?
Svar: De är överlagrade.
2. (3 p) Komplettera den givna koden så att resultatet av anropet $inc[äpa", mus"]$ blir $[bqb", nvt"]$
Svar:

```
instance Incrementable a =>
   Incrementable [a] where
      inc l = map inc l
```