Funktionell programmering

Marcus Dicander

2016-02-25

Strukturen på funktionell programmering

1. Introduktion
2. Typer och Typklasser
3. Högre ordningens funktioner
4. Att använda Monader
5. Sammanfattning och fördjupning

Strukturen idag - Introduktion

1. Vad är funktionell programmering
2. Enkel matematik i GHCI
3. Enkla operationer på listor och strängar i Haskell
4. Funktioner
5. Pattern matching, Guards
6. Rekursion och svansrekursion
7. Lat evaluering, oändliga listor

Vad är funktionell programmering

- En programmeringsstil där den grundläggande operationen är funktionsanrop
- En vidareutveckling av lambdakalkylen från 1930-talet (Alonzo Church)
- Funktionella språk: Lisp, Scheme, Erlang, Haskell, F♯, Clojure

Less is more..

- Inga variabler, ingen initialisering, ingen uppdatering.
- Inga datastrukturer som möjliggör förändringsbart state. Inga föränderingsbara arrayer (Finns i monader, föreläsning 4).
- Inga for-loopar - iteration med rekursion (idag) och högre ordingens funktioner (föreläsning 3)

More is more...

- Full referential transparency - i en fil eller ett metodanrop betyder ett namn alltid samma sak
- Inga sidoeffekter - En funktion som anropas med samma värde två gånger ger alltid samma resultat
- Enkel parallelliserbarhet

Inslag av funktionell programmering i de vanligaste språken

- Java8 har Streams med forEach (map), filter, reduce (foldl) och lambda (föreläsning 3)
- Javascript (ES6)
- C++11 (parallellprogrammeringen, lambdas), C♯
- Python2/Python3 modulerna itertools och functional
- Perl6, pugs - första implementationen av Perl6 skrevs i Haskell (men Rakudo är ledande idag)

Hello World

```
main::IO()
main =
    putStrLn "Hello World!"
```

Glasgow Haskell Compiler

```
$ghc hello.hs
[1 of 1] Compiling Main                ( hello.hs, hello.o )
Linking hello ...
$./hello
Hello World!
```

ghci - Glasgow Haskell Compiler Interactive

```
Prelude> 1+2*3
7
Prelude> 16/3
5.333333333333333
Prelude> 16 `div` 3
5
Prelude> div 16 3
5
```

Vad du kan göra med GHCI

- Beräkna aritmetiska uttryck med +,-,*,/,‘div‘,‘mod‘
- Skapa listor och strängar
- Öppna filer med funktioner för att testa dessa

Skapa listor

```
Prelude> let decimals = [1,4,1,5,9,2,6,5]
Prelude> let firstTen = [1..10]
Prelude> firstTen
[1,2,3,4,5,6,7,8,9,10]
Prelude> let notSeven = [1..6]++[8..10]
```

Skapa fler listor

```
Prelude> let surprise = "Jag är en lista med Chars"
Prelude> surprise
"Jag \228r en lista med Chars"
Prelude> putStrLn surprise
Jag är en lista med Chars
Prelude> let grades = ['A'..'E']
Prelude> let name = "Haskell" ++ " " ++ "Curry"
Prelude>
```

Delar av listor och strängar

- head - första elementet, tail - resten av listan
- init - allt utom sista elementet, last - sista elementet
- take n - de n första elementen, drop n - allt efter de första n elementen
- !! n - Det n:te elementet med nollindexering. Skrivs efter listan.
- alla utom sista skrivs före listan: head [1,2,3]

ghci - Nu med en fil

```
Prelude> :load mathematics.hs
Prelude> :l mathematics.hs
[1 of 1] Compiling Main                ( mathematics.hs, interpr
Ok, modules loaded: Main.
*Main> hypothenuse 3 4
5.0
```

Pattern matching

- Undvik guards och if-satser. Om ett beteende följer direkt av ett konkret värde på en parameter, gör ett pattern för det
- Patterns scannas uppifrån och ned. Gotta catch'em all.
- Om du inte bryr dig om ett parametervärde, använd _

mathematics.hs

```
{- Returns the hypotenuse in a triangle with catheti a and b
hypotenuse::Double->Double->Double
hypotenuse a b =
    sqrt (a^2 + b^2)
```

Collatz Conjecture

$$n_{i+1} = \begin{cases} n_i/2 | even(n_i) \\ 3 * n_i + 1 | odd(n_i) \end{cases}$$

Påstående: Serien når 1 $\forall n \in Z^+$

Obevisad, men verifierad upp till $5.4 * 10^{18}$

Rekursion

- Börja med basfall. Vid vilket parametervärde är du klar?
- Fortsätt med att låta funktionen anropa sig själv så att den steg för steg går mot basfallet och bygger upp svaret

Svansrekursion

- Om det rekursiva anropet returnerar direkt utan några kvarhängade operationer från tidigare anrop så är metoden svansrekursiv
- Svansrekursiva anrop kan optimeras bättre av GHC, men se upp med '++'

Oändliga listor, lat evaluering

```
Prelude> let infinite = [1..]
Prelude> take 10 infinite
[1,2,3,4,5,6,7,8,9,10]
Prelude> take 10 oddCollection
[1,3,5,7,9,11,13,15,17,19]
Prelude> let primes = [2,3,5,7..]

<interactive>:6:22: parse error on input `..'
```