Haskell

Allikas: Vikipeedia

Haskell on standardiseeritud üldotstarbeline puhtalt funktsionaalne programmeerimiskeel. Oma nime on keel saanud loogiku Haskell Curry järgi.[1]

Haskell toetab laiska väärtustamist, näidiste sobitamist, listikomprehensiooni, tüübiklasse ja tüüpide polümorfismi. Kuna tegemist on puhta funktsionaalse keelega, siis puuduvad Haskelli funktsioonidel kõrvalefektid. Viimaste esitamiseks on eraldi andmetüüp: monaad.[2]

Sisukord

Süntaksist üldiselt

Haskellis skripti kirjutamisel on väga oluline taandamine. Näiteks funktsiooni keha ei tohi olla funktsiooni nimega samal kaugusel ekraani vasakust servast. Kui taanded pole korralikult vormistatud, siis kuvatakse vastavasisuline veateade ning programmi ei saa käivitada.

Kuna Haskellis puuduvad tsüklid, siis nende asemel tuleb kasutada rekursiooni. Ebaõnnestunud arvutuse märgiks on nn *bottom*, mida tähistatakse tagurpidi T-tähega (⊥). *Bottom* tekib programmi täitmisaegse vea või lõpmatu arvutuse korral.

Kommentaari algust tähistavad Haskellis kaks järjestikkust sidekriipsu (--). Näiteks

Haskell

Faililaiendid	.hs, .lhs
Paradigma	funktsionaalne, mitte-range, modulaarne
Väljalaskeaeg	1990
Looja	Simon Peyton Jones, Paul Hudak, Philip Wadler ja teised
Viimane väljalase	5.10.1 (22. august 2009)
Tüüpimine	staatiline, tugev, tuletatud
Implementatsioonid	GHC, Hugs, NHC, JHC, Yhc
Dialektid	Helium, Gofer
Mõjutatud keeltest	Lisp ja Scheme, ISWIM, FP, APL, Hope ja Hope+, SISAL, Miranda, ML ja Standard ML, Lazy ML, Orwell, Alfl, Id, Ponder
Mõjutanud keeli	Agda, Bluespec, Clojure, C#, CAL, Cat, Cayenne, Clean, Curry, Epigram, Escher, F#, Factor, Isabelle, Java Generics, LINQ, Mercury, Omega, Perl 6, Python, Qi, Scala, Timber, Visual Basic 9.0
OS	multi-platvormne
Veebileht	haskell.org (http://haskell.org)

```
-- Siin on kommentaar
```

Kommentaar võib olla eraldi real või rea lõpus.

Haskellis on vaikimisi laisk väärtustamine, mis tähendab, et avaldised väärtustatakse ainult siis, kui nende väärtus on tingimata käesoleval arvutuse sammul oluline. Kui aga avaldise väärtus pole hetkel oluline, siis jäetakse väärtustamata avaldis mällu ja vajadusel väärtustatakse hiljem. Haskellis on aga olemas operaator **$!** (dollarimärk ja hüüumärk), mis sunnib agaralt väärtustama. Näiteks deklaratsioonis

```
eelnevateSumma n
  = ($!) eelnevateSumma (n - 1) * n
```

operaator **$!** sunnib väärtustama avaldist **n - 1**.

Hello, world

Järgneb Haskellis kirjutatud Hello world programm (kõik read peale viimase võib ära jätta):

```
module Main where

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

`module Main where` on programmi päis. See sisaldab võtmesõna `module`, protseduuri nime `Main` ja võtmesõna `where`. Rida `main :: IO ()` määrab funktsiooni `main` tüübi, mis antud juhul on tüübile *void* vastav protseduuritüüp (`IO` määrab selle, et tegemist on protseduuritüübiga). Käsk `putStrLn` on lühend väljendist "*put String Line*", mis sisuliselt tähendab, et väljastatakse sõne, millele järgneb reavahetus. Sarnane käsk on `putStr`, kuid sel juhul jäetakse lõpust ära reavahetus.

Listid

Haskellis on olemas listid. Tühja listi tähistab **[]** (nurksulud). Listi võib moodustada näiteks sisestades `[1,2,3,4]`, mille tulemusel luuakse list elementidega 1, 2, 3 ja 4. Sama listi moodustamiseks võib kasutada ka konstruktsiooni `1 : 2 : 3 : 4 : []`, kus elemendid on ühendatud koolonitega ja viimane on tühja listi tähis, mis märgib listi lõppu. Seda listi saab moodustada ka aritmeetilise jada abil: [1 .. 4], kus 1 tähistab esimest elementi ja 4 viimast elementi ning jada samm on vaikimisi 1.

Sõnesid käsitletakse Haskellis ka kui liste. Näiteks "Tere, maailm!" on list, mille elementideks on selles sõnes sisalduvad sümbolid.

Listikomprehensioon

Haskell toetab ka listikomprehensiooni. Näiteks

```
[x ^ 2 | x <- [0, 0.5 .. 3]]
```

Selle tulemusel väljastatakse list, mille elementideks on arvude 0, 0.5, 1, 1.5, 2, 2.5 ja 3 ruudud. `x ^ 2` on avaldis ja `x <- [0, 0.5 .. 3]` on generaator. Generaatori parem pool peab alati olema listitüüpi - näites

sisuliselt list [0, 0.5, 1, 1.5, 2, 2.5, 3].

Näidised (näidiste võrdlemine)

Näidis esitab skeemi, millega ette antud avaldist võrreldakse. Näidis esitab vajalikku ehitust. Näiteks, kui oodatav väärtus, mis ette antakse, on mittetühi list, siis saab näidisega **x : xs** kontrollida, et etteantavas listis oleks vähemalt üks element. Kusjuures **x** tähistab listi pead ehk esimest elementi ning **xs** tähistab listi saba ehk ülejäänud elemente alates teisest elemendist. Näiteks listi [5] korral, mis on üheelemendiline, on **x** väärtuseks 5 ja **xs** väärtuseks [].

Kohane näidis ilmestamaks listi ehituse kontrolli oleks järgnev:

```
ymberPoord xs
  = let
      loppu x : xs            -- 1)
        = loppu xs ++ [x]
      loppu _                 -- 2)
        = []
    in
    loppu xs
```

Protseduur `ymberPoord` saab argumendina ette listi. Kutsutakse välja `loppu` argumendiks saadud listil. Kuna on `loppu` on defineeritud kaks korda (**1)** ja **2)**), siis valitakse näidiste sobitamise teel õige. Esimene näidis **1)** sobitub mittetühjade listidega. Teine näidis **2)** on ainult alakriips ehk jokker, millega sobituvad kõik ette antavad listid. Näiteks tühja listi korral sobitamine näidisega **1)** ebaõnnestub, järelikult täidetakse kood, mis järgneb näidisele **2)**.

Näidise **1)** sobitumisel kutsutakse rekursiivselt välja `loppu` argumendiks olnud listi sabale, kusjuures argumendiks olnud listi pea eraldatakse ja lisatakse lõppu. Rekursiivsed pöördumised toimuvad, kuni argumendiks antakse mittetühi list. Tühja listi korral väljastatakse algne (protseduurile **ymberPoord** argumendina ette antud) list.

Avaldiste liigid

Let-avaldis

Let avaldis võimaldab defineerida hulga lisamuutujaid protseduuri sees. Näiteks

```
tehe x
  = let
      ruut = x * x
      veerand = x / 4
    in
    ruut + veerand
```

See funktsioon tagastab etteantud argumendi veerandi ja ruudu summa. *Let*-avaldise puhul on olulised võtmesõnad **let** ja **in** ning pärast võtmesõna **in** peab järgnema avaldis.

Lambdaavaldis

Lambdaavaldis on näiteks järgnev:

```
viieKordne = \ x -> 5 * x
```

See võtab argumendiks arvu ning tagastab selle arvu viiekordsena. Näiteks, kui argumendiks on 6, tagastab see lambdaavaldis arvu 30. Interaktiivses keskkonnas jõuab sama tulemuseni sisestades

```
(\ x -> 5 * x) 6
```

Lambdaavaldist alustav kaldkriips sarnaneb kreeka tähestiku tähe lambdaga - sellest tõenäoliselt ka nimi "lambdaavaldis".

Valikuavaldis

Valikuavaldis võrdleb argumenti suurema hulga etteantud näidistega ning neist sobiva leidmisel täidab selle näidise kohta käivad käsud ning järgmiseid näidiseid enam ei võrdle.

```
faktoriaal n
  = case compare n 0 of
      GT
        -> ($!) faktoriaal (n - 1) * n
      EQ
        -> 1
      _
        -> error "Argument peab olema mittenegatiivne!"
```

Olulised võtmesõnad on **case** ja **of**. Argumendiks antud arvu võrreldakse nulliga (`compare 0 n`). Kui **n** on nullist suurem, siis kutsutakse rekursiivselt sama funktsiooni uuesti välja ühe võrra väiksemal argumendil. Kui **n** on nulliga võrdne, siis tagastatakse arv 1. Kui argument on negatiivne, siis kuvatakse veateade ("Argument peab olema mittenegatiivne!").

Sisend ja väljund

Haskellis on ka vahendid standardväljundist lugemiseks ning sinna kirjutamiseks. Illustreerimiseks toome järgneva näite:

```
dialoog
  = do
      putStr "Sisesta palun oma nimi: "
      nimi <- getLine
      putStrLn ("Tere, " ++ nimi ++ ". Meeldiv tutvuda!")
```

See funktsioon küsib kasutajalt nime ning tervitab teda seejärel viisakalt nimepidi. Kasutatud on selle funktsiooni koostamisel **do-süntaksit**, mis võimaldab kirjeldada mitmeid tegevusi järjest.

Haskell sisend/väljund käske[3]

- **putChar** - kirjutab standardväljundisse sümboli
- **putStr** - kirjutab standardväljundisse sõne
- **putStrLn** - kirjutab standardväljundisse sõne koos reavahetusega
- **getChar** - loeb standardsisendist sümboli
- **getLine** - loeb standardsisendist rea
- **getContents** - loeb terve faili korraga sisse ühe sõnena

Interaktiivse keskkonna käske

- **:h** - abi (kuvab käskude loendi)
- **:cd** - kaustapuus liikumiseks
- **:i** - kuvab info konkreetse käsu/andmetüübi vms kohta
- **:l** - laadib etteantud failist mooduli interaktiivsesse keskkonda
- **:m** - laadib etteantud standardse mooduli (nt Data.List listidega toimetamiseks)
- **:show modules** - näitab hetkel laetud mooduleid
- **:t** - kuvab etteantud avaldise andmetüübi

Viited

1. ↑ Haskell Wiki: Introduction (http://haskell.org/haskellwiki/Introduction)
2. ↑ Haskell Wiki: Monad (http://www.haskell.org/haskellwiki/Monad)
3. ↑ [1] (http://www.haskell.org/tutorial/io.html)