Programmeerimiskeeled

Kursuse läbinu:

- omandab ülevaatlikud teadmised erinevatest programmeerimise paradigmadest;
- oskab lahendada lihtsaid programmeerimise ülesandeid funktsionaalse programmeerimise abil, sealhulgas saab aru ja oskab kasutada kõrgemat järku polümorfseid funktsioone;
- oskab mitme-paradigma keeles kasutada koos funktsionaalse, imperatiivse ja objekt-orienteeritud paradigma tehnikaid.

Administrativia

- Vastutab
  - Varmo Vene (varmo.vene@ut.ee)
- Loengud
  - Kalmer Apinis (kalmera@ut.ee)
  - Vesal Vojdani (vesal@ut.ee)
- Praksid
  - Kalmer Apinis
  - Raul-Martin Rebane (raul-martin.rebane@ut.ee)
  - Mari Liis Velner (mariliisvelner@gmail.com)
  - Hiie Vill (hiievill@gmail.com)
  - Simmo Saan (simmo.saan@gmail.com)
- Loe pikemalt: courses.cs.ut.ee/2017/PK

Lisamaterjalid

1. “Learn You a Haskell for Great Good!” (2011)
   - \+ Väga hea raamat intuitsiooni saamiseks
   - \- Pole põhjalik ega täielik
2. “Real World Haskell” (2008)
   - \+ Põhjalik ja selge!
   - \- Mahukas!
3. “Sissejuhatus Funktsionaalsesse Programmeerimisse” (2010)
   - \+ Eestikeelne
   - \- Osati liiga detailne, osati liiga piiratud

(Haskelli on vahepeal uuendatud! Raamatud on kohati vananenud!)

- Kui teil on probleeme praksidega, lugege esimest raamatut!
- Kui teid hakkab Haskell huvitama, lugege teist!
- Kui esimesest/teisest raamatust ei saa aru, lugege kolmandat!

Miks uued programmeerimiskeeled?

- Keel C on vähemalt sama võimas, ükskõik mis teine programmeerimiskeel! (järeldub Church-Turingi teesist)
- Vastus: komponentide ja algoritmide taaskasutus! Keegi ei kirjuta programme “nullist”!
  - Näiteks: Algoritmid peavad töötama erinevate andmestruktuuride ja tüüpidega ning võimaldama sisendandmete vajaduspõhist arvutamist.

Funktsionaalne Programmeerimine (FP)

- FP motivatsioon
  - Probleem: võimalused on välja arendamata ja üksteisega konfliktis
  - Võimaluste lisamise asemel peaks üldistama olemasolevaid!
  - Aritmeetilised operatsioonid (funktsioonid) on möödapääsmatud!
- Tüüpimise motivatsioon
  - Probleem: Harva(?) esinevad vead erijuhtudest.
  - Võimaldab vältida vigu ja anda edasi kompileerimisaegset infot.
- Puhta FP motivatsioon
  - Selleks et uurida keerulisi struktuure, peab olema võimalus neid keelata!
  - Näide: hajus ja paralleelne arvutus
- Laisa FP motivatsioon
  - Teoreetiliselt paindlikum kui agar väärtustamine (FP magistriaine).
  - Näide: lõpmatute listidega arvutamine

$\Longrightarrow$ Haskell

Loe lisaks: RWH, eessõna

Haskell

- Haskell erineb Pythonist ja Javast väga palju. Varasemad oskused Pythonist või Javast ei ole otse rakendtavad!
- Seetõttu on Haskelli targem õppida kui matemaatikat/algebrat, mitte kui programmeerimist.
- Väga tähtis on, et te töötaksite juba algusest peale kaasa. Alustame väga lihtsate programmidega ja jõuame alles kursuse lõpuks mingile arvestatavale tasemele.

- Haskelli programm koosneb definitsioonidest
  - Näiteks, loome faili test.hs:

```
a = 40.0
b = 30.0
c = sqrt (a^2 + b^2)
```

- Definitsioone saab lugeda interaktiivsesse keskkonda:

```
> stack ghci test.hs
```

- ... ja siis käivitada

```
*Main> c
50.0
```

- Mida see programm arvutab?
- Mis juhtub, kui muuta definitsioonide järjekorda?

Loe lisaks: RWH, peatükk 1; LYaH, peatükk 2, Starting Out

Tüübid

- Igal defineeritaval nimel on tüüp. Enamasti ei pea tüüpe juurde kirjutama, kuid dokumenteerimise eesmärgil on seda siiski soovitatav aegajalt teha.
- Näiteks:

```
a, b, c :: Float
a = 40
b = 30
c = sqrt (a^2 + b^2)
```

  või

```
a :: Float
a = 40
b :: Float
b = 30
c :: Float
c = sqrt (a^2 + b^2)
```

Loe lisaks: LYaH, peatükk 3, Types and Typeclasses, Believe the type

Funktsioonide defineerimine

- Funktsioone defineeritakse samuti võrdusmärgiga. Võetakse abiks formaalsed parameetrid.
  - näide:

```
pyth :: Float -> Float -> Float
pyth x y = sqrt (x^2 + y^2)
```

- Haskell järgib matemaatilist notatsiooni, kuid on erandeid:
  - "$f(x, y)$" asemel kirjutame "$f\ x\ y$"
- Funktsiooni tüüp on "α -> β", kus α on sisendi tüüp ja β tulemuse tüüp

Faktoriaali näide

- If-avaldisel põhinev faktoriaal

```
fact1 :: Int -> Int
fact1 n = if n==0 then 1 else n * fact1 (n-1)
```

- `fact1` väärtustamine

```
fact1 2
==> if 2 == 0 then 1 else 2 * fact1 (2-1)
==> 2 * fact1 1
==> 2 * (if 1 == 0 then 1 else 1 * fact1 (1-1))
==> 2 * (1 * fact1 0)
==> 2 * (1 * (if 0 == 0 then 1 else 0 * fact1 (0-1)))
==> 2 * (1 * 1)
==> 2
```

- Näidiste sobitamisel põhinev faktoriaal

```
fact2 :: Int -> Int
fact2 0 = 1
fact2 n = n * fact2 (n-1)
```

- Valvuritel põhinev faktoriaal ...

```
fact3 :: Int -> Int
fact3 n
  | n==0      = 1
  | otherwise = n * fact3 (n-1)
```

- Valvuritel põhinev faktoriaal mis ei lähe tsükklisse

```
fact4 :: Int -> Int
fact4 n
  | n == 0 = 1
  | n >= 1 = n * fact4 (n-1)
```

- Akumulaatorit kasutav, where-konstruktsiooniga

```
fact5 :: Int -> Int
fact5 n = fact5' 1 n
       where fact5' a 0 = a
             fact5' a m = fact5' (a*m) (m-1)
```

- Akumulaatorit kasutav, let-konstruktsiooniga

```
fact6 :: Integer -> Integer
fact6 n =
  let fact6' = \ a n -> case n of
                          0 -> a
                          _ -> fact6' (a*n) (n-1)
  in fact6' 1 n
```

- product funktsiooni ja jada kasutav faktoriaal

```
fact7 :: Int -> Int
fact7 n = product [1..n]
```

Loe lisaks: LYaH, peatükk 4

Vindi ülekeeramine ...

- Püsipunktikombinaatori abil defineeritud faktoriaal

```
fact9 :: Integer -> Integer
fact9 = fixedPt f
  where f g 0     = 1
        f g n     = n * g (n-1)
        fixedPt f = g where g = f g
```

- “The Evolution of a Haskell Programmer”
  - https://www.willamette.edu/~fruehr/haskell/evolution.html

Programmeerimise paradigmad

Saab jagada kaheks:

- imperatiivsed e. mis operatsioone teha
  - Lisaks jaotatakse: protseduuraalsed ja objektorienteeritud
- deklaratiivsed e. lahenduse (tõe) kirjeldamine
  - Lisaks jaotatakse: funktsionaalne ja loogiline

Erinevused:

- Imperatiivne kasvas välja protsessori käsustiku abstaheerimisest.
- Deklaratiivne kasvas välja matemaatikast.

Matemaatik/loogik tahab mõelda

- algebralisteststruktuuridest nagu rühmad, monoidid, ring jne.
- funktsioonidest, hulkadest ja relatsioonidest

Paljud protsessori instruktsioonid pole lihtsalt modelleeritavad – keerulised erijuhud.

Poolt ja vastuargumendid

- Võimaldab kirjeldada garantiisid arvutuste kohta ja keelab sul neid rikkuda …
- … aga see võib tähendada lisatööd.

- Enamasti pole kontrolli, kuidas programm tõlgitakse masinkoodi.
- Programmide efektiivsus ei ole mitte alati piisav.
- Aja- ja mälukeerukus võib olla raskesti ennustatav.

Niisiis:

- Tuleb endale selgeks teha kõik võimalused.
- Kasutada erinevaid võimalusi seal, kus see annab eelise.
- Olla valmis tulevikuks. (näiteks 15 aasta pärast)

Funktsionaalsed keeled

FP lühiajalugu

- Kombinaatorloogika (M. Schönfinkel 1924, H. Curry 1927)
- Lambda-arvutus (A. Church 1936)
- Lisp (J. McCarthy 1958)
- ML ja polümorfne tüübisüsteem (R. Milner 1978)
- Hope, Sasl, Miranda, . . . (1980 – 85)
- Haskell (1988)

Haskelli jalugu

- 1987 loodi Haskell'i komitee
- 1988 esimene keelekirjeldus (v. 1.0)
- 1999 Haskell98 (Standard Haskell)
- 2010 Haskell 2010
- 2016 GHC 8 (suur standardteegi puhastus)