Kava

* 1. loeng
  Admin
  Motivatsioon

I: definitsioonid, kasutamine
I: funktsioonide defineerimine
I: faktoriaali näited

Administrativia

- Loengud
  - K10-12, r1008, Varmo ja Kalmer
- Praksid
  - E10-12, r2030, Danel Ahman?
  - E12-14, r2030, Karoliine Holter
  - T10-12, r2006, Aiden Madisson
  - R10-12, r2005, Kalmer Apinis?
- Avalik veeb: `https://courses.cs.ut.ee/2024/fp`
  - loengute materjalid, videod
  - praktikumide ja kodutööde materjalid
- Kursuse privaat-veeb: Moodle
  - testid, tagasiside
  - kodutööde esitamine, automaathindamine

Hinde kujunemine

- 0p..50% $\rightarrow$ F, 51%..60% $\rightarrow$ E, 61%..70% $\rightarrow$ D, ...
  - ümardame üles (70.1 $\rightarrow$ 71 $\rightarrow$ C)
- Kontrolltööd (2*31p, praksi ajal + sessi. ajal)
- Testid (14*1p, moodle)
- Kodutööd (12*2p, moodle)
- (Active) Quiz (boonuspunktid, 0.5p, moodle)

Funktsionaalprogrammeerimine

Kursuse läbinu:

- oskab selgitada λ-arvutuse põhimõisteid ning analüüsida vastavaid programme;

Funktsionaalprogrammeerimine

Kursuse läbinu:

- oskab selgitada λ-arvutuse põhimõisteid ning analüüsida vastavaid programme;
- oskab selgitada FP põhimõisteid;

Funktsionaalprogrammeerimine

Kursuse läbinu:

- oskab selgitada λ-arvutuse põhimõisteid ning analüüsida vastavaid programme;
- oskab selgitada FP põhimõisteid;
- oskab lahendada lihtsaid ülesandeid funktsionaalse programmeerimise abil; sealhulgas oskab kasutada kõrgemat järku polümorfseid funktsioone;

Funktsionaalprogrammeerimine

Kursuse läbinu:

- oskab selgitada λ-arvutuse põhimõisteid ning analüüsida vastavaid programme;
- oskab selgitada FP põhimõisteid;
- oskab lahendada lihtsaid ülesandeid funktsionaalse programmeerimise abil; sealhulgas oskab kasutada kõrgemat järku polümorfseid funktsioone;
- oskab kirjeldada funktsionaalse paradigma eeliseid ja puudusi.

Miks uued programmeerimiskeeled?

- Keel C on vähemalt sama võimas, ükskõik mis teine programmeerimiskeel! (järeldub Church-Turingi teesist)

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”! Mida lihtsam on erinevaid teeke omavahel ühendada, seda parem.

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”! Mida lihtsam on erinevaid teeke omavahel ühendada, seda parem.
- Vastus: korrektsuse tõestamise võimalus. Meil on palju koodi aga me ei saa seda usaldada.

Motivatsioon

Koodi taaskasutusel on abiks

- paindlikud modulaarsuse võimalused
- selge arusaam piirangutest (eeltingimused/garantiid)

Motivatsioon

Koodi taaskasutusel on abiks

- paindlikud modulaarsuse võimalused
- selge arusaam piirangutest (eeltingimused/garantiid)

Korrektsuse tõestamisel on abiks

- Keel on täpselt defineeritud.
- Range tüübisüsteem.
- Ilmutatud viidatavus ehk saame ignoreerida ebaolulist.
  - Puhas FP – funktsiooni tagastusväärtus sõltub ainul argumentidest.
- Baaskonstruktsioonide hulk on väike.

Motivatsioon

Koodi taaskasutusel on abiks

- paindlikud modulaarsuse võimalused
- selge arusaam piirangutest (eeltingimused/garantiid)

Korrektsuse tõestamisel on abiks

- Keel on täpselt defineeritud.
- Range tüübisüsteem.
- Ilmutatud viidatavus ehk saame ignoreerida ebaolulist.
  - Puhas FP – funktsiooni tagastusväärtus sõltub ainul argumentidest.
- Baaskonstruktsioonide hulk on väike.

$\Longrightarrow$ Õpime Funktsionaalset Programmeerimist!

1. loeng

Idris vs. Haskell

- Haskell on küps ja praktikas kasutatav FP keel.
- Haskelli fookus ei ole (enam) FP algõpe.
- Idris on väga sarnane Haskellile aga pole (veel) küps.

Idris vs. Haskell

- Haskell on küps ja praktikas kasutatav FP keel.
- Haskelli fookus ei ole (enam) FP algõpe.
- Idris on väga sarnane Haskellile aga pole (veel) küps.
- Idris 2-l on põnevad omadused, mida tahame hiljem näidata.

Selleks, et kursusel kasutada üht keelt, kasutame Idris 2-e.
(Aga kommenteerime, millised erinevused on Haskellis.)

Idris

- Raamat: *Type-Driven development with Idris*, Edwin Brady
- Idrise (ka Haskelli) programm sisaldab definitsioone; igal defineeritaval nimel on tüüp.
  - Näiteks, loome faili test.idr:

```
a : Double
a = 40.0

b : Double
b = 30.0

c : Double
c = sqrt (a*a + b*b)
```

Idris

- Raamat: *Type-Driven development with Idris*, Edwin Brady
- Idrise (ka Haskelli) programm sisaldab definitsioone; igal defineeritaval nimel on tüüp.
  - Näiteks, loome faili test.idr:

```
a : Double
a = 40.0

b : Double
b = 30.0

c : Double
c = sqrt (a*a + b*b)
```

- Definitsioone saab lugeda interaktiivsesse keskkonda:

```
> idris2 test.idr
```

Idris

- Raamat: *Type-Driven development with Idris*, Edwin Brady
- Idrise (ka Haskelli) programm sisaldab definitsioone; igal defineeritaval nimel on tüüp.
  - Näiteks, loome faili test.idr:

```
a : Double
a = 40.0

b : Double
b = 30.0

c : Double
c = sqrt (a*a + b*b)
```

- Definitsioone saab lugeda interaktiivsesse keskkonda:

```
> idris2 test.idr
```

- ... ja siis väärtustada

```
*Main> c
50.0
```

Funktsioonide defineerimine

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

```
pyth : Double → Double → Double
pyth x y = sqrt (x*x + y*y)
```

  - NB! Slaididel on → aga failis kirjutame ->

Funktsioonide defineerimine

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

    ```
    pyth : Double → Double → Double
    pyth x y = sqrt (x*x + y*y)
    ```

  - NB! Slaididel on → aga failis kirjutame ->

- Funktsiooni tüüp on "$\alpha \to \beta$", kus α on sisendi ja β tulemuse tüüp.
  - Kahe argumendiga funktsioon on "`a → (b → c)`",
    ehk "`a → b → c`".

Funktsioonide defineerimine

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

    ```
    pyth : Double → Double → Double
    pyth x y = sqrt (x*x + y*y)
    ```

  - NB! Slaididel on → aga failis kirjutame ->

- Funktsiooni tüüp on "$\alpha \to \beta$", kus α on sisendi ja β tulemuse tüüp.
  - Kahe argumendiga funktsioon on "`a → (b → c)`",
    ehk "`a → b → c`".

- Funktsiooni argumendid kirjutatakse funktsiooni järele:

  ```
  Main> pyth 3 4
  5.0
  ```

Faktoriaali näide

- If-avaldisel põhinev faktoriaal

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

Faktoriaali näide

- If-avaldisel põhinev faktoriaal

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

- `fact1` väärtustamine

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
```

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)
```

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)
⟹ if False then 1 else 2 * fact1 (2-1)
```

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)
⟹ if False then 1 else 2 * fact1 (2-1)
⟹ 2 * fact1 (2-1)
```

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)
⟹ if False then 1 else 2 * fact1 (2-1)
⟹ 2 * fact1 (2-1)
⟹ 2 * fact1 1
```

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

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

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

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

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

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)
⟹ if False then 1 else 2 * fact1 (2-1)
⟹ 2 * fact1 (2-1)
⟹ 2 * fact1 1
⟹ 2 * (if 1 == 0 then 1 else 1 * fact1 (1-1))
⟹ 2 * (if False then 1 else 1 * fact1 (1-1))
⟹ 2 * (1 * fact1 (1-1))
⟹ 2 * (1 * fact1 0)
⟹ 2 * (1 * (if 0 == 0 then 1 else 0 * fact1 (-1)))
⟹ 2 * (1 * (if True then 1 else 0 * fact1 (0-1)))
```

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)
⟹ if False then 1 else 2 * fact1 (2-1)
⟹ 2 * fact1 (2-1)
⟹ 2 * fact1 1
⟹ 2 * (if 1 == 0 then 1 else 1 * fact1 (1-1))
⟹ 2 * (if False then 1 else 1 * fact1 (1-1))
⟹ 2 * (1 * fact1 (1-1))
⟹ 2 * (1 * fact1 0)
⟹ 2 * (1 * (if 0 == 0 then 1 else 0 * fact1 (-1)))
⟹ 2 * (1 * (if True then 1 else 0 * fact1 (0-1)))
⟹ 2 * (1 * 1)
```

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)
⟹ if False then 1 else 2 * fact1 (2-1)
⟹ 2 * fact1 (2-1)
⟹ 2 * fact1 1
⟹ 2 * (if 1 == 0 then 1 else 1 * fact1 (1-1))
⟹ 2 * (if False then 1 else 1 * fact1 (1-1))
⟹ 2 * (1 * fact1 (1-1))
⟹ 2 * (1 * fact1 0)
⟹ 2 * (1 * (if 0 == 0 then 1 else 0 * fact1 (-1)))
⟹ 2 * (1 * (if True then 1 else 0 * fact1 (0-1)))
⟹ 2 * (1 * 1)
⟹ 2
```

- Näidiste/mustri sobitamisel põhinev faktoriaal

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

- Näidiste/mustri sobitamisel põhinev faktoriaal

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

- `with`-konstruktsioonil põhinev faktoriaal . . .

```
fact3 : Int → Int
fact3 n with (n==0)
          _ | True  = 1
          _ | False = n * fact3 (n-1)
```

- Näidiste/mustri sobitamisel põhinev faktoriaal

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

- `with`-konstruktsioonil põhinev faktoriaal . . .

```
fact3 : Int → Int
fact3 n with (n==0)
          _ | True  = 1
          _ | False = n * fact3 (n-1)
```

- alternatiivne `with`-konstruktsioonil põhinev faktoriaal . . .

```
fact4 : Int → Int
fact4 n with (n)
          _ | 0 = 1
          _ | _ = n * fact4 (n-1)
```

- Akumulaatorit kasutav, where-konstruktsiooniga

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

  - `fact5'` taane ja joondus on oluline!

- Akumulaatorit kasutav, where-konstruktsiooniga

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

  - `fact5'` taane ja joondus on oluline!

- Akumulaatorit kasutav, let-konstruktsiooniga

```
fact6 : Integer → Integer
fact6 n =
  let
    fact6' : Integer → Integer → Integer
    fact6' = λ a, n ⇒ case n of
                        0 ⇒ a
                        _ ⇒ fact6' (a*n) (n-1)
  in fact6' 1 n
```

- Akumulaatorit kasutav, where-konstruktsiooniga

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

  - `fact5'` taane ja joondus on oluline!

- Akumulaatorit kasutav, let-konstruktsiooniga

```
fact6 : Integer → Integer
fact6 n =
  let
    fact6' : Integer → Integer → Integer
    fact6' = λ a, n ⇒ case n of
                        0 ⇒ a
                        _ ⇒ fact6' (a*n) (n-1)
  in fact6' 1 n
```

- `product` funktsiooni ja listi kasutav faktoriaal

```
fact7 : Int → Int
fact7 n = product [1..n]
```