Funktsionaalprogrammeerimine

Administrativia

- Loengud
  - K10-12, Δ-1008, Varmo ja Kalmer
- Praksid
  - E10-12, Δ-2030, Kalmer Apinis
  - E12-14, Δ-2030, Karoliine Holter
- Avalik veeb: `https://courses.cs.ut.ee/2023/fp`
  - loengute materjalid
  - praktikumide ja kodutööde materjalid
- Kursuse privaat-veeb: Moodle
  - loengutestid
  - kodutööde esitamine

Hinde kujunemine

- 0p..50p $\rightarrow$ F, 51p..60p $\rightarrow$ E, 61p..70p $\rightarrow$ D, ...
  - ümardame üles (70.1 $\rightarrow$ 71 $\rightarrow$ C)
- Kontrolltööd (2*32p, miinimumtase 2*10p, praksi ajal + sessi. ajal)
- Kodutööd (12*3p, moodle)
- Lisaks projektiülesanne+esitlus (6 boonuspunkti)
- Active Quiz ja Tagasiside (iga testi eest 0.5 boonuspunkt, moodle)

Funktsionaalprogrammeerimine

Kursuse läbinu:

- oskab selgitada λ-arvutuse põhimõisteid ning analüüsida λ-terme;
- 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.

Active Quiz!

1. loeng

Miks uued programmeerimiskeeled?

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

Miks uued programmeerimiskeeled?

- 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

Taaskasutusel on abiks

- head modulaarsuse võimalused
- koodi eeltingimused ja garantiid

Motivatsioon

Taaskasutusel on abiks

- head modulaarsuse võimalused
- koodi eeltingimused ja garantiid

Korrektsuse tõestamisel on abiks, et ...

- Keel on täpselt defineeritud.
- Baaskonstruktsioonide hulk on väike.
- Range tüübisüsteem.
- Ilmutatud viidatavus ehk selge, mis on ebaoluline.

$\Longrightarrow$ Õpime Funktsionaalset Programmeerimist?

Miks mitte 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.

Miks mitte 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.)

Näide: Faktoriaal

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

fact väärtustamine

```
fact 2
```

Näide: Faktoriaal

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

fact väärtustamine

```
fact 2
⟹ 2 * fact (2-1)
```

Näide: Faktoriaal

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

fact väärtustamine

```
fact 2
```

$\Longrightarrow$ `2 * fact (2-1)`

$\Longrightarrow$ `2 * fact 1`

Näide: Faktoriaal

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

fact väärtustamine

```
fact 2
```

$\Longrightarrow$ `2 * fact (2-1)`

$\Longrightarrow$ `2 * fact 1`

$\Longrightarrow$ `2 * (1 * fact (1-1))`

Näide: Faktoriaal

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

fact väärtustamine

```
    fact 2
⟹  2 * fact (2-1)
⟹  2 * fact 1
⟹  2 * (1 * fact (1-1))
⟹  2 * (1 * fact 0)
```

Näide: Faktoriaal

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

fact väärtustamine

```
fact 2
⟹ 2 * fact (2-1)
⟹ 2 * fact 1
⟹ 2 * (1 * fact (1-1))
⟹ 2 * (1 * fact 0)
⟹ 2 * (1 * 1)
```

Näide: Faktoriaal

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

fact väärtustamine

```
    fact 2
⟹  2 * fact (2-1)
⟹  2 * fact 1
⟹  2 * (1 * fact (1-1))
⟹  2 * (1 * fact 0)
⟹  2 * (1 * 1)
⟹  2 * 1
```

Näide: Faktoriaal

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

fact väärtustamine

```
    fact 2
⟹  2 * fact (2-1)
⟹  2 * fact 1
⟹  2 * (1 * fact (1-1))
⟹  2 * (1 * fact 0)
⟹  2 * (1 * 1)
⟹  2 * 1
⟹  2
```

Näide: Faktoriaal

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

fact väärtustamine

```
    fact 2
⟹  2 * fact (2-1)
⟹  2 * fact 1
⟹  2 * (1 * fact (1-1))
⟹  2 * (1 * fact 0)
⟹  2 * (1 * 1)
⟹  2 * 1
⟹  2
```

Active Quiz!

FP määratlemine

Programmeerimiskeeli on kombeks jagada paradigmadeks:

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

Enamus keeli on multi-paradigma keeled — saab kasutada erinevaid stiile.

FP määratlemine

Programmeerimiskeeli on kombeks jagada paradigmadeks:

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

Enamus keeli on multi-paradigma keeled — saab kasutada erinevaid stiile.

Erinevused (intuitsioon):

- Imperatiivne on lähedane protsessori käsustikule.
- Deklaratiivne on lähedane matemaatikale/loogikale.

FP määratlemine

Programmeerimiskeeli on kombeks jagada paradigmadeks:

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

Enamus keeli on multi-paradigma keeled — saab kasutada erinevaid stiile.

Erinevused (intuitsioon):

- Imperatiivne on lähedane protsessori käsustikule.
- Deklaratiivne on lähedane matemaatikale/loogikale.

Matemaatik/loogik tahab mõelda kõrgemal tasemel:

- algebralistest struktuuridest nagu rühmad, monoidid, ringid jne.
- funktsioonidest, hulkadest ja relatsioonidest

FP määratlemine

Programmeerimiskeeli on kombeks jagada paradigmadeks:

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

Enamus keeli on multi-paradigma keeled — saab kasutada erinevaid stiile.

Erinevused (intuitsioon):

- Imperatiivne on lähedane protsessori käsustikule.
- Deklaratiivne on lähedane matemaatikale/loogikale.

Matemaatik/loogik tahab mõelda kõrgemal tasemel:

- algebralistest struktuuridest nagu rühmad, monoidid, ringid jne.
- funktsioonidest, hulkadest ja relatsioonidest

Paljud protsessori instruktsioonid pole (matemaatikule) intuitiivsed.
Näiteks, kui palju on Javas: `Math.abs(-2147483648)`? Active Quiz!

Funktsionaalne keel

- Funktsioon: valem mille järgi arvutada konkreetse väärtuse.
- Ilma funktsioonideta (meetodite, protseduurideta) ei saa/taha!

  Java: `Math.max(4, 7)`
  Python: `max(4,7)`
  Idris: `max 4 7`

- Sõna „funktsioon“ asemel kasutatakse ka *abstraktsioon*.
  (Kuna enamasti ei huvita, kuidas `max` on implementeeritud.)

Funktsionaalne keel

- Funktsioon: valem mille järgi arvutada konkreetse väärtuse.
- Ilma funktsioonideta (meetodite, protseduurideta) ei saa/taha!

  Java: `Math.max(4, 7)`
  Python: `max(4,7)`
  Idris: `max 4 7`

- Sõna „funktsioon“ asemel kasutatakse ka *abstraktsioon*.
  (Kuna enamasti ei huvita, kuidas `max` on implementeeritud.)
- FP eelistab olemasolevate vahendite üldistamist.

  Näiteks: Samamoodi, kuidas programmeerimiskeeles saab käsitleda andmeid, peab saama käsitleda ka alamprogramme.

  $\Longrightarrow$ `max 3` saab võtta kõigist listi `[2,3,4]` elementidest (3. loeng)

```
map (max 3) [2,3,4] == [max 3 2, max 3 3, max 3 4] == [3,3,4]
```

Puhas keel

- Puhas ehk kõrvaltoimevaba — funktsiooni kutse tulemus sõltub ainult parameetrite väärtustest.

  $\Longrightarrow$ iga funktsiooni saab testida teistest eraldi

- Mittepuhtaid funktsioone (nagu juhuarvude genereerimine) pole võimalik funktsioonidena defineerida

  aga saab *modelleerida* kasutades puhtaid funktsioone.

Tugevalt ja staatiliselt tüübitud

- Mida rohkem programmeerimiskeel lubab seda parem?
  - Ei! Näiteks, Portable Document Format (1993) on suuresti vähendatud võimalustega PostScript (1982).
- Piiramise üks võimalus on *tüübisüsteemiga*.
  - Tüüp kirjeldab (ja kitsendab) programmi alamosa ja tüübikontroll otsustab, kas neid osi tohib niimoodi kombineerida.
  - Näiteks: Kui `a` on tüüpi `Int` siis seda saab anda `max` argumendiks.
  - Testimise vajadus mingil määral väiksem?

Tugevalt ja staatiliselt tüübitud

- Mida rohkem programmeerimiskeel lubab seda parem?
  - Ei! Näiteks, Portable Document Format (1993) on suuresti vähendatud võimalustega PostScript (1982).
- Piiramise üks võimalus on *tüübisüsteemiga*.
  - Tüüp kirjeldab (ja kitsendab) programmi alamosa ja tüübikontroll otsustab, kas neid osi tohib niimoodi kombineerida.
  - Näiteks: Kui `a` on tüüpi `Int` siis seda saab anda `max` argumendiks.
  - Testimise vajadus mingil määral väiksem?
- Staatiline kontroll — kompileerimise käigus
- Tugevalt tüübitud — väljastatakse kohe veateade.