Administrativia

- Loengud
  - K10-12, r1020, Varmo ja Kalmer
- Konsultatsioon, vastuvõtuaeg
  - K12-14, r3022, Kalmer
- Praksid
  - E10-12, r2048/6, Kalmer Apinis
  - E12-14, r2006, Karoliine Holter
  - T10-12, r2006, Kalmer Apinis
  - R10-12, r2047, Danel Ahman
- Avalik veeb: `https://courses.cs.ut.ee/2026/fp`
  - loengute materjalid, videod
  - FP Õpik (mustand)
  - praktikumide ja kodutööde materjalid
- Kursuse privaat-veeb: Moodle
  - testid, tagasiside
  - kodutööde esitamine, automaathindamine
- Suhtlus tudengitega: Zulip

Hinde kujunemine

- 0p..50% $\rightarrow$ F, 51%..60% $\rightarrow$ E, 61%..70% $\rightarrow$ D, ...
  - ümardame üles (70.1 $\rightarrow$ 71 $\rightarrow$ C)
- Kontrolltööd (2*33p max)
  - 6. ja 16. nädalal + jaanuaris
- Väikesed kodutööd (12*0.5p, moodle)
- Suured kodutööd (2*10p, moodle)
- Testid ja tagasiside (16*0.5p, moodle)
- Lisapunktid (?*0.5p, boonus)

Õpiväljundid

Kursuse läbinu:

- kirjutab funktsionaalseid programme funktsionaalses keeles (Idris2), kasutades baasväärtusi, liste, rekursiooni ja mustrisobitust;
- kasutab kõrgemat järku ja parameetriliselt polümorfseid funktsioone ning defineerib uusi andmetüüpe ülesannete lahendamiseks;
- rakendab liideseid, monaade ja sisend-väljundit ning laiska väärtustamist ja lõpmatuid struktuure;
- rakendab sõltuvaid tüüpe programmide korrektsuse tagamiseks;
- selgitab ja analüüsib lambda-arvutuse põhimõisteid;
- selgitab funktsionaalse programmeerimise teoreetilisi aluseid;
- võrdleb funktsionaalset paradigmat teiste paradigmadega, tuues välja selle eelised ja puudused.

Õpiväljundid

Kursuse läbinu:

- kirjutab funktsionaalseid programme funktsionaalses keeles (Idris2), kasutades baasväärtusi, liste, rekursiooni ja mustrisobitust;
- kasutab kõrgemat järku ja parameetriliselt polümorfseid funktsioone ning defineerib uusi andmetüüpe ülesannete lahendamiseks;
- rakendab liideseid, monaade ja sisend-väljundit ning laiska väärtustamist ja lõpmatuid struktuure;
- rakendab sõltuvaid tüüpe programmide korrektsuse tagamiseks;
- selgitab ja analüüsib lambda-arvutuse põhimõisteid;
- selgitab funktsionaalse programmeerimise teoreetilisi aluseid;
- võrdleb funktsionaalset paradigmat teiste paradigmadega, tuues välja selle eelised ja puudused.

Aga miks seda kõike vaja on?!

Miks funktsionaalprogrammeerimine?!

- FP stiilis saab kirjutada nii Pythonis, C++s, Javas jne. (Harjutame Idrist, et saada paremaks Pythonis.)
- Uued põnevad lahendused pärinevad FP-st või sisaldavad tihti FPd. Näiteks Elm ja Svelte.
- Keeled ja teegid tulevad ja lähevad. FP on inspiratsioon arvutiteaduse tipptegijatele peaaegu 70 aastat.
- ...

Miks λ-arvutus?!

Kontekstiks, λ-arvutus Pythonis:

```
>>> (lambda x: x+7)(2*3)
13
```

Uurime λ-arvutust, kuna

- see on ajalooliselt olnud FP alus;
- tahame teada, mida üldse on võimalik arvutiga teha.

Tehisaru?

$$\text{Tudeng} + \text{ChatGPT} = \text{Väga tark}$$

$$\sim$$

$$\text{Tudeng} + \text{Elon Musk} = \text{Väga rikas}$$

Kui sa kasutad tehisaru abi, küsi endalt järgnevat.

- Mis on minu osa? Milleks mina kasulik olen?
- Kas tehisaru kasutamine aitab jõuda eesmärkideni?
- Miks peaks keegi andma sulle 20x+ ChatGPT kuutasu?

See kursus on õppimiseks. Kui sa ei taha õppida, siis miks sa siia tulid?

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)

Korrektsuse tõestamisel on abiks

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

$\Longrightarrow$ Õpime Funktsionaalset Programmeerimist!

FP keelte tugevused

Populaarsed FP keeled:

- (Python, C++, Java, ...: küps keel, tööstuses kasutusel)
- Haskell: laisk, tüübiklassidega, küps keel, tööstuses kasutusel
- OCaml: agar, moodulitega, küps keel, tööstuses kasutusel
- Rocq: mõeldud tõestamiseks, küps keel, saab programme genereerida
- Lean4: mõeldud tõestamiseks, saab programmeerida
- ...

Millist õppimiseks valida? Tahame keelt, mis

1. "sunnib" FP stiili kasutama
2. võimalikult selge süntaksiga
3. võimaldab sõltuvate tüüpidega progemist ja tõestamist

Pole ideaalset keelt, kus kõike mugavalt teha saab! Idris on hea kompromiss.

Idris

- Raamat: *Type-Driven development with Idris*, Edwin Brady
- Raamat: *FP õpik (mustand)*, Kalmer Apinis, Varmo Vene
- Idrise 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
- Raamat: *FP õpik (mustand)*, Kalmer Apinis, Varmo Vene
- Idrise 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
- Raamat: *FP õpik (mustand)*, Kalmer Apinis, Varmo Vene
- Idrise 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 ->

- 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, 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]
```