Funkcijsko programiranje

Tema: Uvod. Upoznavanje s Haskellom. Primjeri definiranja jednostavnih funkcija.

10. listopada 2018.

predavač: doc. dr. sc. Slobodan Jelić

asistent: Luka Borozan, mag. math.

1. Uvod
   - Funkcije
   - Funkcijsko programiranje
   - Svojstva Haskella
   - Povjesni razvoj
   - Primjeri programa u Haskellu

2. Upoznavanje s Haskellom
   - Primjeri
   - Primjena funkcija
   - Pravila dodjeljivanja imena funkcijama
   - Pravila uvlačenja koda i komentari

Funkcije I

- Što je funkcija?
- U matematici:

Definicija

Funkcija $f : D \to K$ je preslikavanje koje svakom elementu domene D pridružuje točno jedan element kodomene K. Funkcija je zadana domenom, kodomenom i pravilom preslikavanja.

- Primjer funkcije `udvostruci`:

```
udvostruci x = x + x

main = print (udvostruci 2)
```

- prilikom primjene funkcije na određeni argument, rezultat se dobije zamjenom naziva argumenata u tijelu funkcije s vrijednostima argumenata

Funkcije II

- Kako će se evaluirati poziv `udvostruci (udvostruci 2)`?
- Zaključak:
  - redoslijed primjene funckija u prethodnom primjeru ne utječe na rezultata, ali utječe na broj koraka u evaluaciji

Funkcijsko programiranje I

Definicija

Funkcijsko programiranje je koncept programiranja u kojem je osnovna metoda primjena funkcije na argumente.

Zadatak

Računanje sume prvih n prirodnih brojeva.

- Proceduralni pristup:

```
1  brojac ← 0
2  ukupno ← 0
3  repeat
4          brojac ← brojac + 1
5          ukupno ← ukupno + brojac
6     until brojac = n
```

Funkcijsko programiranje II

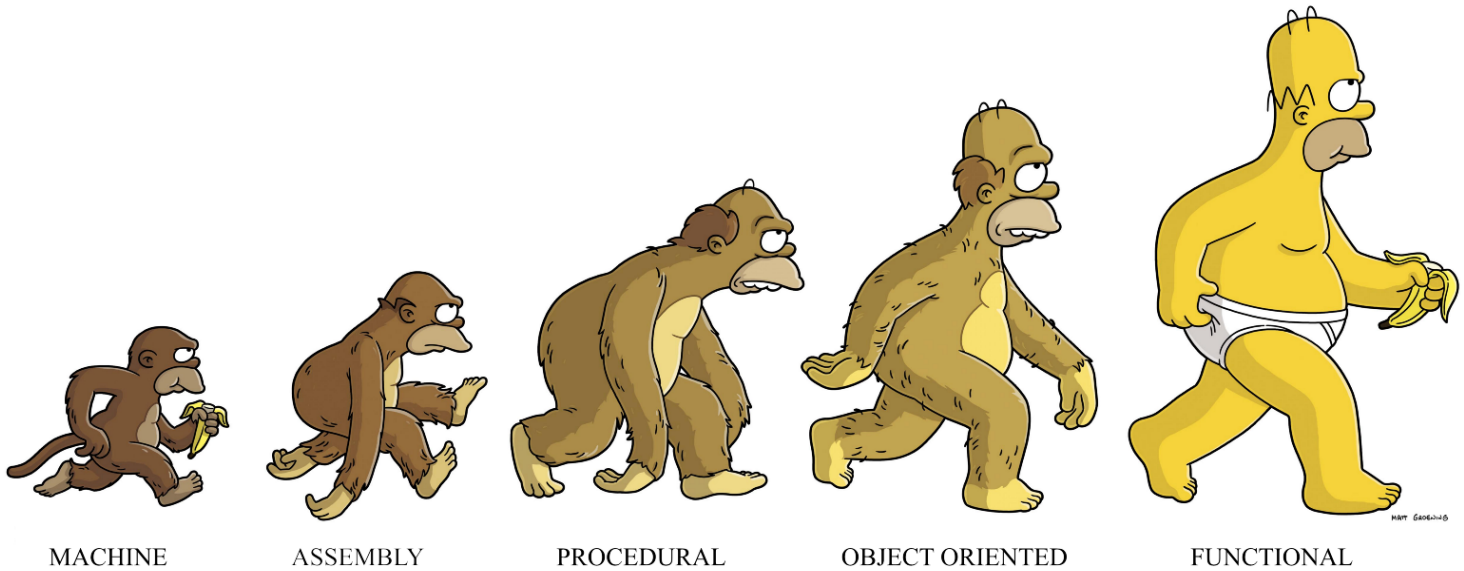
- programski jezici u kojima je osnovna metoda izmjena spremljenih vrijednosti zovu se *imperativni jezici*
- Funkcijski pristup:

```
sum[1..5]
```

- većina imperativnih jezika ne podržava funkcijski pristup (primjerice, spremanje funkcija u strukture podataka kao što su liste)
- u ovom kolegiju proučavat ćemo programski jezik **Haskell** koji podržava funkcijski pristup

Svojstva Haskella I

- jasni i koncizni programi
  - programski jezik visoke razine
  - programski kod je kraći i do nekoliko puta u odnosu na programske jezike s kojima ste se susreli



Svojstva Haskella II

- moćan sustav tipova
- komprehenzija listi
- rekurzivne funkcije
- funkcije višeg reda
- programiranje s efeftima: monade
- lijene evaluacije
- rezoniranje o programima

Povjesni razvoj

- (1930-e), Alonzo Church - razvoj lambda - računa (eng. lambda calculus) i matematičke teorije funkcija
- (1950-e) John McCarthy - Lisp (LISt Processor) prvi funkcijski jezik za programiranje, zadržao dodjeljivanje vrijednosti varijablama kao jedan od osnovnih koncepata
- (1960-e) Peter Landin - ISWIM (”If you See What I Mean”) prvi čist programski jezik za funkcijsko programiranje
- (1970-e) John Backus - FP (”Functional Programming”) prvi uzima u obzir koncepte funkcija višeg reda i rezoniranja o programima
- (1970-e) Robin Milner i ostali - ML (”Meta Language”) prvi moderan programski jezik za funkcijsko programiranje koji uzima u obzir polimorfne tipove i zaključivanje o tipovima
- (1987) pokrenut razvoj Haskella kao standardnog programskog jezika za funcksijko programiranje s lijenim evaluacijama
- (2003) stabilna verzija Haskella

Primjeri programa u Haskellu I

Primjer

Računanje n-tog neparnog broja.

Primjeri programa u Haskellu II

Primjer

Računanje sume elemenata liste duljine n.

Primjeri programa u Haskellu III

```
suma [] = 0
suma (x : xs) = x + suma xs
```

- ukoliko je lista prazna, sume je jednaka 0
- ukoliko je lista neprazna, suma je jedanaka zbroju prvog elementa i sume preostalih elemenata
- kažemo da je funkcija `suma` zadana je **rekurzivno** i završit će u konačno mnogo koraka
- svaka funkcija ima tipove argumenata i rezultata što je izvedeno na osnovu definicija funkcije

Primjeri programa u Haskellu IV

Primjer

Sortiranje pomoću Quick-Sort algoritma.

Primjeri programa u Haskellu V

```
qs :: [Int] -> [Int]
qs [] = []
qs (x : xs) = qs manjijedx ++ [x] ++ qs vecix
    where manjijedx = [a | a <- xs, a <= x ]
          vecix = [b | b <- xs, b > x ]
```

- ++ je operator koji spaja dvije liste
- `where` je ključna riječ koja uvodi lokalne definicije listi `manjijedx` i `vecix`

Primjeri I

- dohvati prvi element neprazne liste

```
head [1,2,3,4,5]
```

- ukloni prvi element neprazne liste

```
tail [1,2,3,4,5]
```

- dohvati n-ti element neprazne liste

```
[1,2,3,4,5] !! 2
```

- dohvati prvih n elemenata neprazne liste

```
take 3 [1,2,3,4,5]
```

- ukloni prvih n elemenata neprazne liste

```
drop 3 [1,2,3,4,5]
```

- izračunaj duljinu liste

```
length [1,2,3,4,5]
```

Primjeri II

- izračunaj sumu elemenata liste brojeva

```
sum [1,2,3,4,5]
```

- izračunaj produkt elemenata liste brojeva

```
product [1,2,3,4,5]
```

- spoji dvije liste

```
[1,2,3] ++ [4,5]
```

- obrni redoslijed elemenata liste

```
reverse [1,2,3,4,5]
```

Primjena funkcija I

- u matematici:

$$f(x, y) + xy$$

- u Haskellu:

$$f\,x\,y + x * y$$

- **primjena funkcije** ima veći prioritet od svih drugih operatora

Matematika	Haskell
$f(x)$	`f x`
$f(x, y)$	`f x y`
$f(g(x))$	`f (g x)`
$f(x, g(x))$	`f x (g y)`
$f(x)g(x)$	`f x * g y`

Pravila dodjeljivanja imena funkcijama I

- imena funkcija i njihovih argumenata moraju početi s malim slovom, mogu nastaviti s bilo kojim slovom, brojem, podvlakom, jednostrukim navodnikom
- ključne riječi:

```
case class data default deriving do else
if import in infix infixl infixr instance
let module newtype of then type where
```

- uobičajeno se nazivim listi daje sufiks `s`, primjerice: `ns`, `xs`

Pravila uvlačenja koda i komentari

- svaka definicija unutar skripte mora početi u istom stupcu

```
a = b + c
  where
    b = 1
    c = 2
d = a * 2
```

- definicije se grupiraju prema uvlakama
- linijski komentari počinju `--`
- višelinijski (blokovski) komentari su između {- i -}
- izbjegavanje tabulatornih razmaka (”tabova”)