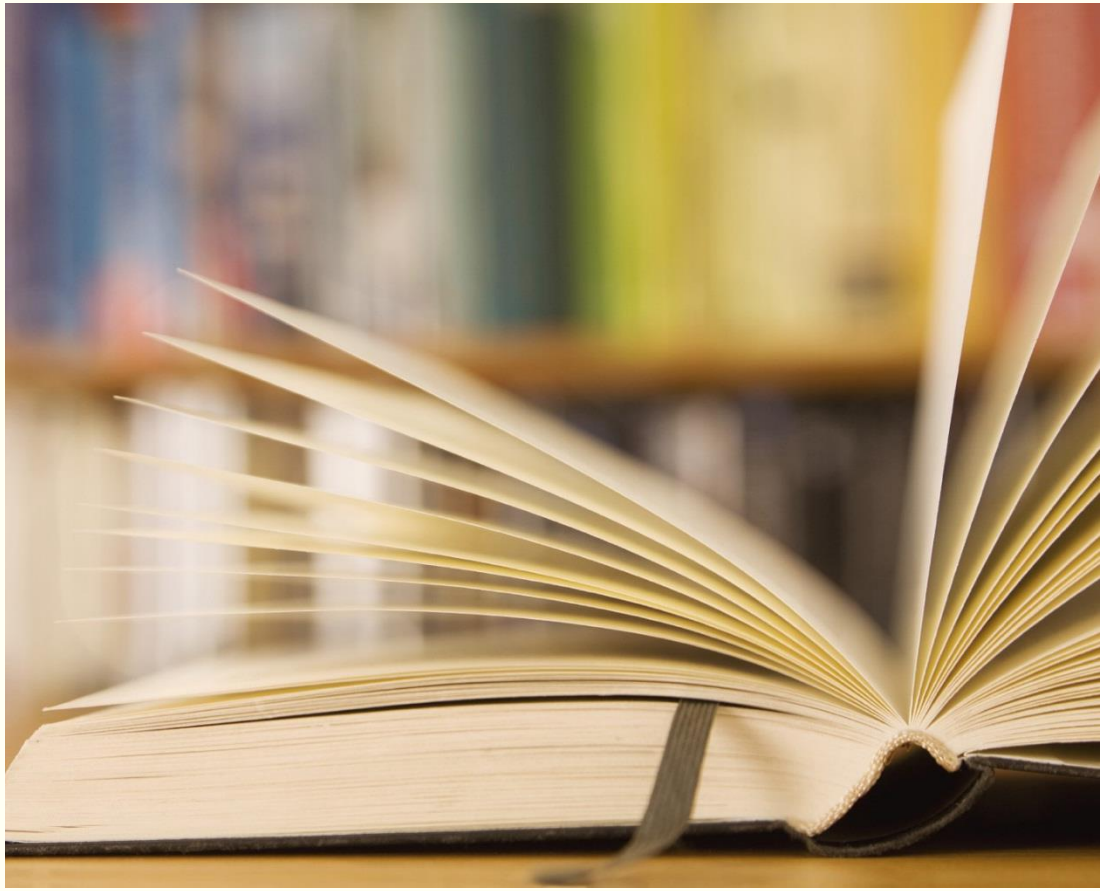
FUNKCIONALNO PROGRAMIRANJE

Uvod u Haskell



Funkcije

- U Haskell-u, funkcija je presilkavanje koje uzima jedan ili više argumenata i vraća rezultat.
  - Šta sve čini funkciju?

$$double\ x \quad = \quad x + x$$

- Izračunavanje rezultata;

$$\begin{array}{ll} & double\ 3 \\ = & \quad \{\ \text{applying } double\ \} \\ & 3 + 3 \\ = & \quad \{\ \text{applying } +\ \} \\ & 6 \end{array}$$

Funkcije

Redoslijed izvršenja?

$$
\begin{aligned}
& \mathit{double}\ (\mathit{double}\ 2) \\
= & \quad \{ \text{ applying the inner } \mathit{double} \ \} \\
& \mathit{double}\ (2+2) \\
= & \quad \{ \text{ applying } + \ \} \\
& \mathit{double}\ 4 \\
= & \quad \{ \text{ applying } \mathit{double} \ \} \\
& 4+4 \\
= & \quad \{ \text{ applying } + \ \} \\
& 8
\end{aligned}
$$

$$
\begin{aligned}
& \mathit{double}\ (\mathit{double}\ 2) \\
= & \quad \{ \text{ applying the outer } \mathit{double} \ \} \\
& \mathit{double}\ 2 + \mathit{double}\ 2 \\
= & \quad \{ \text{ applying the first } \mathit{double} \ \} \\
& (2+2) + \mathit{double}\ 2 \\
= & \quad \{ \text{ applying the first } + \ \} \\
& 4 + \mathit{double}\ 2 \\
= & \quad \{ \text{ applying } \mathit{double} \ \} \\
& 4 + (2+2) \\
= & \quad \{ \text{ applying the second } + \ \} \\
& 4+4 \\
= & \quad \{ \text{ applying } + \ \} \\
& 8
\end{aligned}
$$

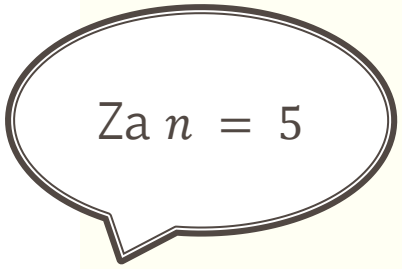
Fukcionalno programiranje

- Stil programiranja u kojem je osnovni metod izračunavanja primjena funkcije na argumente.
- Izračunavanje sume brojeva od 0 do n.

  -- imperativni jezici

```
count := 0
total := 0
repeat
    count := count + 1
    total := total + count
until
    count = n
```

$count$	:=	0
$total$	:=	0
$count$	:=	1
$total$	:=	1
$count$	:=	2
$total$	:=	3
$count$	:=	3
$total$	:=	6
$count$	:=	4
$total$	:=	10
$count$	:=	5
$total$	:=	15



Fukcionalno programiranje

- Izračunavanje sume brojeva od 0 do n. – Haskell
  - [..] – lista brojeva od 1 do n;
  - sum – suma elemenata liste;

$$sum\ [1\,..\,n]$$

Za $n = 5$

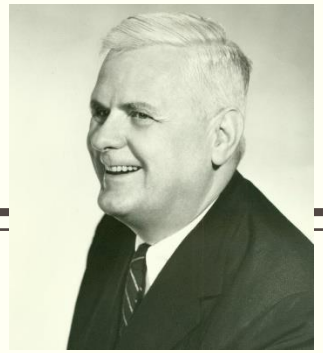
$$\begin{array}{ll} & sum\ [1\,..\,5] \\ = & \quad \{\ \text{applying}\ [\ ..\ ]\ \} \\ & sum\ [1,2,3,4,5] \\ = & \quad \{\ \text{applying}\ sum\ \} \\ & 1+2+3+4+5 \\ = & \quad \{\ \text{applying}\ +\ \} \\ & 15 \end{array}$$

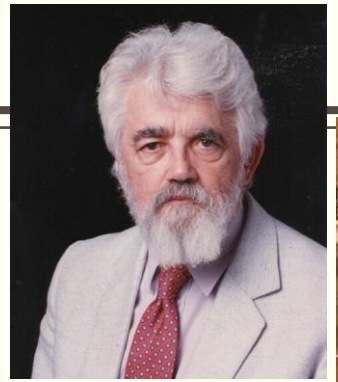
- Imperativni jezici vs Haskell;

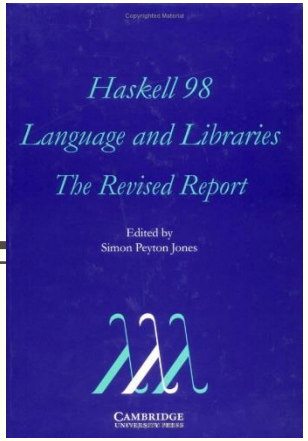
Haskell

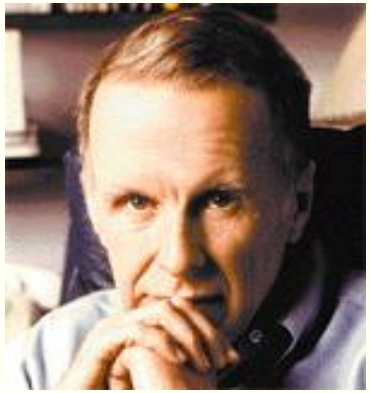
- Koncizni programi;
- Moćan sistem tipova;
- Zadavanje listi;
- Rekurzija;
- Funkcije višeg reda;
- Monade;
- Lijeno izračunavanje;

Istorijski razvoj

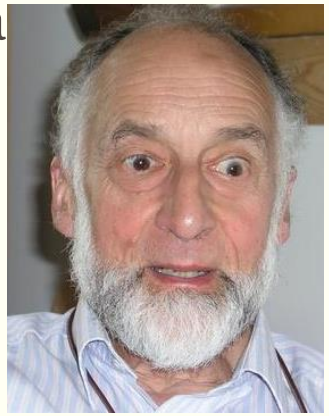








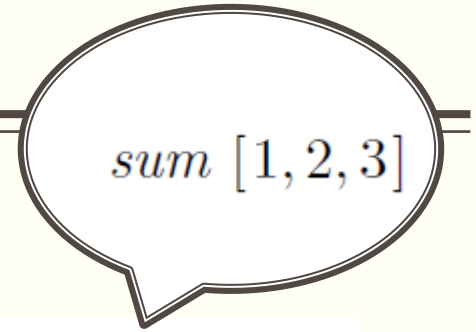
- 1930. Alonzo Čerč – lambda račun;
- 1950. Džon MekKarti* – Lisp (LISt Processor);
- 1960. Peter Landin - ISWIM (If you See What I Mean);
- 1970. Džon Bakus* - FP (Functional Programing);
- 1970. Robin Milner* - ML (Meta Language);
- 1970.-1980. Dejvid Turner - brojni lijeni funkcionalni programski jezici Miranda
- 1987. internacionalni komitet istraživača – Haskell;
- 2003. Haskell Report;



*ACM Turning Award;

Haskell
A Purely Functional Language

Par primjera u Haskell-u



- Funkcija *sum*

$$\begin{aligned} sum\ [\,] &= 0 \\ sum\ (x : xs) &= x + sum\ xs \end{aligned}$$

- Tip funkcije

$$Num\ a \Rightarrow [a] \rightarrow a$$

$$\begin{aligned} & sum\ [1,2,3] \\ = & \quad \{\ \text{applying } sum\ \} \\ & 1 + sum\ [2,3] \\ = & \quad \{\ \text{applying } sum\ \} \\ & 1 + (2 + sum\ [3]) \\ = & \quad \{\ \text{applying } sum\ \} \\ & 1 + (2 + (3 + sum\ [\,])) \\ = & \quad \{\ \text{applying } sum\ \} \\ & 1 + (2 + (3 + 0)) \\ = & \quad \{\ \text{applying } +\ \} \\ & 6 \end{aligned}$$

Par primjera u Haskell-u

- Funkcija *qsort*

$$
\begin{array}{lcl}
qsort\ [\,] & = & [\,] \\
qsort\ (x : xs) & = & qsort\ smaller +\!\!+ [x] +\!\!+ qsort\ larger \\
 & & \textbf{where} \\
 & & \quad smaller = [a \mid a \leftarrow xs, a \leqslant x] \\
 & & \quad larger = [b \mid b \leftarrow xs, b > x]
\end{array}
$$

- ++ - spaja dvije liste u jednu;
- **where** – ključna riječ kojom se uvode lokalne definicije;

Par primjera u Haskell-u

$$
\begin{aligned}
& qsort\ [x] \\
= & \quad \{\ \text{applying } qsort\ \} \\
& qsort\ [\,] +\!\!+ [x] +\!\!+ qsort\ [\,] \\
= & \quad \{\ \text{applying } qsort\ \} \\
& [\,] +\!\!+ [x] +\!\!+ [\,] \\
= & \quad \{\ \text{applying } +\!\!+\ \} \\
& [x]
\end{aligned}
$$

Par primjera u Haskell-u

$$
\begin{aligned}
& qsort\ [3,5,1,4,2] \\
= & \quad \{\ \text{applying } qsort\ \} \\
& qsort\ [1,2] +\!\!+ [3] +\!\!+ qsort\ [5,4] \\
= & \quad \{\ \text{applying } qsort\ \} \\
& (qsort\ [] +\!\!+ [1] +\!\!+ qsort\ [2]) +\!\!+ [3] +\!\!+ (qsort\ [4] +\!\!+ [5] +\!\!+ qsort\ []) \\
= & \quad \{\ \text{applying } qsort, \text{ above property}\ \} \\
& ([] +\!\!+ [1] +\!\!+ [2]) +\!\!+ [3] +\!\!+ ([4] +\!\!+ [5] +\!\!+ []) \\
= & \quad \{\ \text{applying } +\!\!+\ \} \\
& [1,2] +\!\!+ [3] +\!\!+ [4,5] \\
= & \quad \{\ \text{applying } +\!\!+\ \} \\
& [1,2,3,4,5]
\end{aligned}
$$

Par primjera u Haskell-u

- Tip funkcije

$$qsort :: Ord\ a \Rightarrow [a] \rightarrow [a]$$

Zadaća:

- Knjiga – 1.7 Exercises (strana 17)

HASKELL – PRVI KORACI

Hugs sistem

- Sistem Hugs najčešće korištena implementacija Haskell 98.
- Interpreter;
- GHC, nhc98, UHC, Yhc, jhc, Ihc...

Instalacija GHC na nekim OS

- Windows
  - Najpre instalirajte chocolatey (najverovatnije sa admin ovlašćenjima, powershell):
    - https://chocolatey.org/install
  - Nakon toga instalirajte GHC naredbom (najverovatnije sa admin ovlašćenjima, powershell ili cmd):
    - choco install -y haskell-dev
- Ubuntu
  - Najpre ažurirajte pakete:
    - apt-get update
  - Zatim instalirajte:
    - apt-get install haskell-platform -y

Pokretanje GHCI

- Pokretanje Hugs sistema;

```
> ghci

GHCi, version X: http://www.haskell.org/ghc/  :? for help

Prelude>
```

- prompt > - interpreter je spreman za izračunavanje izraza;

Pokretanje GHCI

Prioritet?
Asocijativnost?

```
> 2+3*4
14

> (2+3)*4
20

> sqrt (3^2 + 4^2)
5.0
```

```
> 2-3
-1

> 7 `div` 2
3

> 2 ^3
8
```

Standardna biblioteka Prelude

- Funkcije za rad sa listama;
- Vraća prvi element liste (glavu liste)

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

- Uklanja prvi element liste (vraća rep liste);

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

Standardna biblioteka Prelude

- Vraća n-ti element liste (broji se od 0);

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

- Vraća prvih n elemenata liste;

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

Standardna biblioteka Prelude

- Uklanja prvih n elemenata liste;

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

- Računa dužinu liste (broj elemenata koji se nalaze u listi);

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

- Računa zbir elemenata liste; Tip?

```
> sum [1,2,3,4,5]
15
```

Standardna biblioteka Prelude

- Računa proizvod elemenata liste; Tip?

```
> product [1,2,3,4,5]
120
```

- Spaja dvije liste u jednu;

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

- Okreće redoslijed elementa liste;

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

Standardna biblioteka Prelude

- Moguće greške;

```
> 1 `div` 0
Error

> head []
Error
```

Primjena funkcija

- Matematika:

```
f(a,b) + c d
```

- Haskell:

```
f a b + c*d
```

```
f a + b
```

Prioritet?

(f a) + b, a ne f (a + b).

Primjena funkcija

Mathematics	Haskell
$f(x)$	$f\ x$
$f(x, y)$	$f\ x\ y$
$f(g(x))$	$f\ (g\ x)$
$f(x, g(y))$	$f\ x\ (g\ y)$
$f(x)\, g(y)$	$f\ x * g\ y$

Haskell skripte

- Nove funkcije se definišu u haskell skriptama.
- Skripta – tekstualni fajl koji sadrži defincije funkcija.
- Ekstenzija .hs;
- Prva skripta u Haskell-u:
  - Pokrenemo editor;
  - Prekucama sljedeće dvije definicije;
  - Sačuvamo kao test.hs;

```
double x     = x + x

quadruple x = double (double x)
```

```
> :load test.hs
```

```
> quadruple 10
40

> take (double 2) [1,2,3,4,5,6]
[1,2,3,4]
```

Haskell skripte

- U test.hs dodati sljedeće definicije funkcija;

```
factorial n = product [1..n]

average ns = sum ns `div` length ns
```

- U terminalu:

```
> :reload

> factorial 10
3628800

> average [1,2,3,4,5]
3
```

GHCI komande

Skraćenice?

Command	Meaning
:*load name*	load script *name*
:*reload*	reload current script
:*edit name*	edit script *name*
:*edit*	edit current script
:*type expr*	show type of *expr*
:?	show all commands
:*quit*	quit Hugs

Konvencija imenovanja

- Imena funkcija počinju malim slovom, iza kojeg može da slijedi proizvoljno mnogo malih slova, velikih slova, cifara, donjih crta i jednostrukih navodnika.

```
myFun    fun1    arg_2    x'
```

- Ključne riječi:

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

- Imena lista se završavaju sufiksom s.

```
xs    ns    nss
```

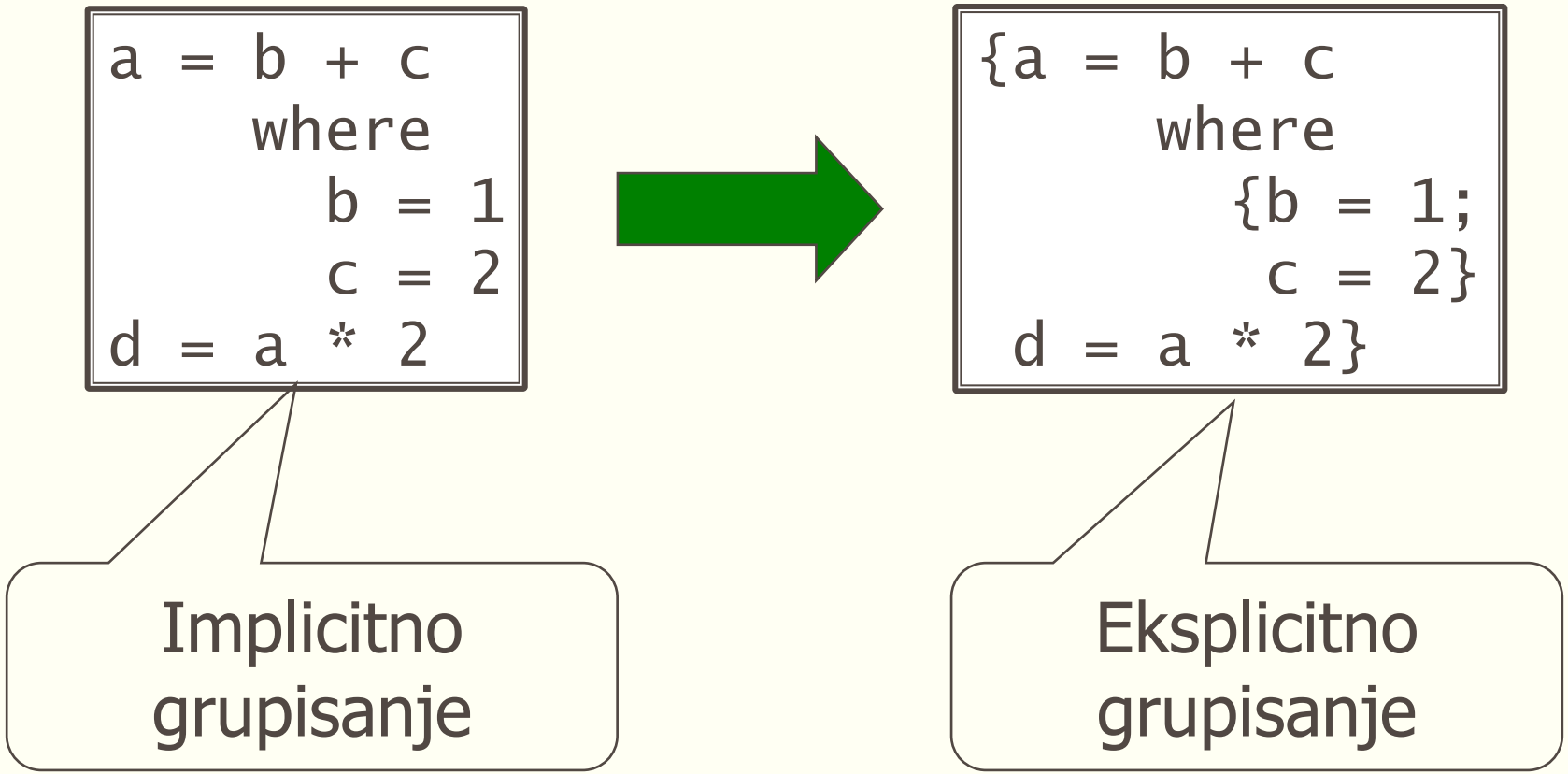
Poravnanje koda u Haskell-u

```
a = 10
b = 20
c = 30
```

```
a = 10
 b = 20
c = 30
```

```
 a = 10
b = 20
 c = 30
```

Poravnanje koda u Haskell-u



Komentari

- Jednolinijski komentari;

```
-- Factorial of a positive integer:
factorial n  =  product [1..n]
-- Average of a list of integers:
average ns  =  sum ns `div` length ns
```

- Višelinijski komentari;

```
{-
double x      =  x + x
quadruple x   =  double (double x)
-}
```

Ugnježdeni komentari

Zadaci:

1. Postaviti zagrade koje odgovaraju redoslijedu izvršenja operacija u Haskell-u.

$$2 \uparrow 3 * 4$$
$$2 * 3 + 4 * 5$$
$$2 + 3 * 4 \uparrow 5$$

2. Funkcija $last$ vraća posljednji element neprazne liste. Koristeći uvedene funkcije iz biblioteke Perlude definisati funkciju $last$.
3. Funkcija $init$ uklanja posljednji element neprazne liste. Koristeći uvedene funkcije iz biblioteke Perlude definisati funkciju $init$.

TIPOVI I KLASE

Haskell

Osnovni koncepti

- Tip je kolekcija poveznih vrijednosti.
  - $Bool$: $True, False$
  - $Bool \rightarrow Bool$
- Notacija
  - v je tipa T

$$v :: T$$

$$e :: T$$

v je vrijednost, e je izraz

  - Npr.

$$\begin{array}{lll} False & :: & Bool \\ True & :: & Bool \\ \neg & :: & Bool \rightarrow Bool \end{array}$$

$$\begin{array}{lll} \neg\ False & :: & Bool \\ \neg\ True & :: & Bool \\ \neg\ (\neg\ False) & :: & Bool \end{array}$$

Osnovni koncepti

- U Haskell-u svaki izraz ima tip, koji se određuje prije izračunavanja izraza u procesu zaključivanje tipa (*type inference)*.

$$\frac{f :: A \to B \qquad e :: A}{f\ e :: B}$$

- Primjer:
  - $\neg\ False\ ::\ Bool\ ?$
  - $\neg\ 3?$
- Haskell je *type-safe* jezik.
  - if $True$ then 1 else $False$ – type error!

Osnovni koncepti

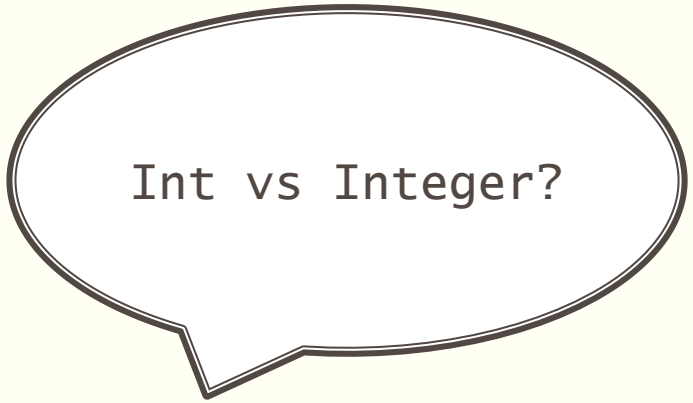
- Provjera tipova u ghci:

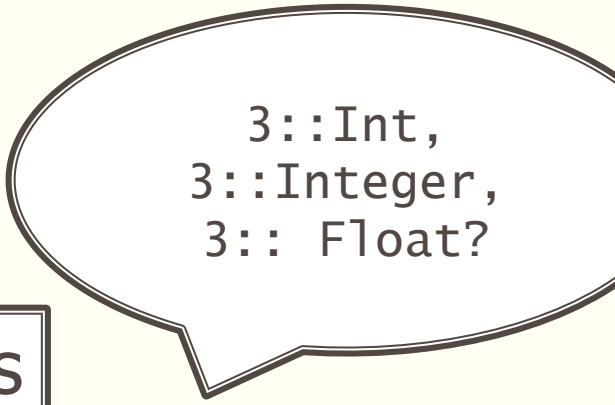
```
> :type not
not :: Bool -> Bool

> not False
True

> :type not False
not False :: Bool

> :type not 3
Error
```

Osnovni tipovi

- Osnovni tipovi podataka u Haskell-u:

`Bool`	- logical values
`Char`	- single characters
`String`	- strings of characters
`Int`	- fixed-precision integers
`Integer`	- arbitrary-precision integers
`Float`	- floating-point numbers





Liste - tipovi

- Lista je niz elementa istog tipa, navedenih unutar [ ], međusobno odvojenih zarezima.
  - [T] je oznaka tipa liste koja sadrži elemente tipa T;

```
[False,True,False] :: [Bool]

['a','b','c','d']  :: [Char]

["One","Two","Three"]  :: [String]
```

  - Dužina liste?
  - [], ['a'], [[]]?

Liste - tipovi

- Tip liste ne sadrži informaciju o dužini liste.

```
[False,True]        :: [Bool]

[False,True,False] :: [Bool]
```

- Šta sve može biti element liste?

```
[['a'],['b','c']] :: [[Char]]
```

- Beskonačne liste;

Torke - tipovi

- Torka je konačan niz komponenti, koje mogu biti različitih tipova, navedenih unutar (), odvojenih zarezima.
  - $(T_1, T_2, \ldots, T_n)$ – i-ta komponenta je tipa T_i.

```
(False,True)      :: (Bool,Bool)

(False,'a',True)  :: (Bool,Char,Bool)

("Yes",'a',True)  :: (String,Char,Bool)
```

  - *arnost* torke;
  - () – prazna torka; dvojka; trojka; ...; n-torka;
  - $(False)$ – nije dozvoljeno!

Torke - tipovi

- Tip torke sadrži informaciju o arnosti torke.

```
(False,True)         :: (Bool,Bool)

(False,True,False) :: (Bool,Bool,Bool)
```

- Šta sve može biti komponenta torke?

```
('a',(False,'b')) :: (Char,(Bool,Char))

(True,['a','b'])  :: (Bool,[Char])
```

- Arnost torke je konačna.

Tipovi funkcija

- Funkcija je preslikavanje elemenata jednog skupa tipova u drugi skup tipova.
  - $T_1 \rightarrow T_2$;

```
not  :: Bool → Bool

even :: Int → Bool

isDigit:: Char→ Bool
```

Tipovi funkcija

- Deklaracija tipa funkcije prije definicije funkcije;


```
add         :: (Int,Int) → Int
add (x,y)    = x+y

zeroto      :: Int → [Int]
zeroto n     = [0..n]
```

Karijeve funkcije

- Funkcije sa više argumenata moguće je definisati koristeći funkciju kao povratnu vrijednost:

add' uzima kao argument cijeli broj x i vraća kao rezultat funkciju add' x. Slično, ova funkcija uzima cijeli broj y i vraća kao rezultat funkciju $x + y$.

```
add'     :: Int → (Int → Int)
add' x y = x+y
```

Karijeve funkcije

```
add  :: (Int,Int) → Int

add' :: Int → (Int → Int)
```

- add i add' imaju isti konačan rezultat, ali add uzima oba argumenta istovremeno, dok add' uzima jedan po jedan argument.
- Funkcije koje uzimaju jedan po jedan argument zovu se Karijeve funkcije, u čast rada Haskell Curry-ja nad ovim funkcijama.

Karijeve funkcije

- Funkcije sa više od dva argumenta mogu se definisati kao Karijeve korištenjem ugnježdenih funkcija kao povratnih vrijednosti:

mult uzima argument x i vraća funkciju mult x, koja slično uzima argument y i vraća funkciju mult x y, koja na kraju uzima argument z i vraća rezultat x*y*z.

```
mult       :: Int → (Int → (Int → Int))
mult x y z = x*y*z
```

Karijeve funkcije

- Zašto su Karijeve funkcije korisne?
  - Karijeve funckije su fleksibilnije nego funkcije nad torkama jer se parcijalnom primjenom Karijevh funkcija mogu dobiti razne korisne funkcije.

```
add' 1 :: Int → Int

take 5 :: [Int] → [Int]

drop 5 :: [Int] → [Int]
```

Karijeve funkcije

- Konvencije u zapisu Karijevih funkcija
  - Strelica → je desno asocijativna.

Int → (Int → (Int → Int))

```
Int → Int → Int → Int
```

  - Primjena funkcija je lijevo asocijativna.

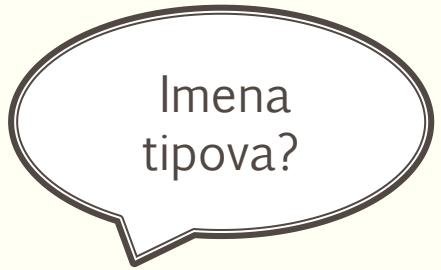
((mult x) y) z

```
mult x y z
```

Polimorfni tipovi

- Funkcija je polimorfna ako se može primjeniti na više različitih tipova.
  - Funkcija *length*

```
> length [1,3,5,7]
4

> length ["Yes","No"]
2

> length [isDigit, isLower, isUpper]
3
```




```
length :: [a] → Int
```

Polimorfni tipovi

- Još neke polimorfne funkcije;

```
fst  :: (a,b) → a

head :: [a] → a

take :: Int → [a] → [a]

zip  :: [a] → [b] → [(a,b)]

id   :: a → a
```

Preopterećeni tipovi

- Aritmetički operator + je preopterećen.

```
> 1 + 2
3

> 1.1 + 2.2
3.3
```

Ograničenje klase tipova

```
(+) :: Num a ⇒ a -> a -> a
```

Preopterećeni tipovi

```
> 1 + 2
3

> 1.0 + 2.0
3.0

> 'a' + 'b'
ERROR
```

a = Int

a = Float

Char is not a numeric type

Preopterećeni tipovi

- Još neke preopterećene funkcije:

$$
\begin{array}{lcl}
(-) & :: & Num\ a \Rightarrow a \rightarrow a \rightarrow a \\
(*) & :: & Num\ a \Rightarrow a \rightarrow a \rightarrow a \\
negate & :: & Num\ a \Rightarrow a \rightarrow a \\
abs & :: & Num\ a \Rightarrow a \rightarrow a \\
signum & :: & Num\ a \Rightarrow a \rightarrow a
\end{array}
$$

Osnovne klase tipova

- Tip - kolekcija poveznih vrijednosti.
- Klasa – kolekcija tipova koji podržavaju određene preoptrećene operacije.
- Eq klasa

$$
\begin{aligned}
(==) &:: a \to a \to Bool \\
(\neq) &:: a \to a \to Bool
\end{aligned}
$$

  - $Bool, Char, String, Int, Integer, Float$
  - Liste, torke;

Funkcije?

```
> False == False
True

> 'a' == 'b'
False

> "abc" == "abc"
True

> [1,2] == [1,2,3]
False

> ('a', False) == ('a', False)
True
```

Osnovne klase tipova

- *Ord* klasa

$$
\begin{array}{lcl}
(<) & :: & a \to a \to Bool \\
(\leqslant) & :: & a \to a \to Bool \\
(>) & :: & a \to a \to Bool \\
(\geqslant) & :: & a \to a \to Bool \\
min & :: & a \to a \to a \\
max & :: & a \to a \to a
\end{array}
$$

  - *Bool, Char, String, Int, Integer, Float*
  - Liste, torke;

```
> False < True
True

> min 'a' 'b'
'a'

> "elegant" < "elephant"
True
```

```
> [1,2,3] < [1,2]
False

> ('a',2) < ('b',1)
True

> ('a',2) < ('a',1)
False
```

Osnovne klase tipova

- *Show* klasa
  - Tipovi koji se mogu konvertovati u String upotrebom metode

$$show \quad :: \quad a \rightarrow String$$

  - $Bool, Char, String, Int, Integer, Float$
  - Liste, torke;

```
> show False
"False"

> show 'a'
"'a'"

> show 123
"123"

> show [1,2,3]
"[1,2,3]"

> show ('a', False)
"('a',False)"
```

Osnovne klase tipova

- *Read* klasa
  - Dualna klasa klasi *Show*

$$read :: String \rightarrow a$$

  - *Bool, Char, String, Int, Integer, Float*
  - Liste, torke;
- :: - razrješava tip rezultata;
  - U praksi iz konteksta;

$$\neg\ (read\ \texttt{"False"})$$

  - Nedefinisan rezultat;

$$\neg\ (read\ \texttt{"hello"})$$

$> read\ \texttt{"False"} :: Bool$
$False$

$> read\ \texttt{"'a'"} :: Char$
$\texttt{'a'}$

$> read\ \texttt{"123"} :: Int$
123

$> read\ \texttt{"[1,2,3]"} :: [Int]$
$[1, 2, 3]$

$> read\ \texttt{"('a',False)"} :: (Char, Bool)$
$(\texttt{'a'}, False)$

Osnovne klase tipova

- *Num* klasa

$$
\begin{array}{lll}
(+) & :: & a \to a \to a \\
(-) & :: & a \to a \to a \\
(*) & :: & a \to a \to a \\
negate & :: & a \to a \\
abs & :: & a \to a \\
signum & :: & a \to a
\end{array}
$$

  - *Int*, *Integer*, *Float*

$$
\begin{array}{l}
> \ 1 + 2 \\
3
\end{array}
$$

$$
\begin{array}{l}
> \ 1.1 + 2.2 \\
3.3
\end{array}
$$

$$
\begin{array}{l}
> \ negate \ 3.3 \\
-3.3
\end{array}
$$

$$
\begin{array}{l}
> \ abs \ (-3) \\
3
\end{array}
$$

$$
\begin{array}{l}
> \ signum \ (-3) \\
-1
\end{array}
$$

Osnovne klase tipova

- $Integral$ klasa

$$
\begin{aligned}
div &:: a \to a \to a \\
mod &:: a \to a \to a
\end{aligned}
$$

$$
\begin{aligned}
&> 7\ `div`\ 2 \\
&3 \\
\\
&> 7\ `mod`\ 2 \\
&1
\end{aligned}
$$

- Funkcije koje za argumente uzimaju cijeli broj i listu ($length, take, drop$) – Int;
  - List.hs

Osnovne klase tipova

- *Fractional* klasa

$$
\begin{array}{lcl}
(/) & :: & a \to a \to a \\
recip & :: & a \to a
\end{array}
$$

$$
\begin{array}{l}
> \ 7.0 \ / \ 2.0 \\
3.5 \\
\\
> \ recip \ 2.0 \\
0.5
\end{array}
$$

Zadaci:

4. Odrediti tipove sljedećih vrijednosti:

$$
\begin{array}{l}
[\texttt{'a'}, \texttt{'b'}, \texttt{'c'}] \\
(\texttt{'a'}, \texttt{'b'}, \texttt{'c'}) \\
[(False, \texttt{'0'}), (True, \texttt{'1'})] \\
([False, True], [\texttt{'0'}, \texttt{'1'}]) \\
[tail, init, reverse]
\end{array}
$$

5. Odrediti tipove sljedećih funkcija:

$$
\begin{array}{lcl}
second\ xs & = & head\ (tail\ xs) \\
swap\ (x, y) & = & (y, x) \\
pair\ x\ y & = & (x, y) \\
double\ x & = & x * 2 \\
palindrome\ xs & = & reverse\ xs == xs \\
twice\ f\ x & = & f\ (f\ x)
\end{array}
$$

Zadaci:

6. Napisati funkcije za konverziju KM u eure, i obrnuto. Kog tipa su napisane funkcije?

Nove funkcije na osnovu postojećih funkcija

7. Napisati funkciju koja provjerava da li je dati karakter cifra.
8. Napisati funkciju koja ispituje da li je dati broj paran.
9. Napisati funkciju koja dijeli listu na n-tom elementu.
10. Napisati funkciju koja vraća recipročnu vrijednost datog broja.

Uslovni izrazi

- **if … then … else**

11. Napisati funkciju koja vraća apsolutnu vrijednost datog broja.

- **Ugnježdeni uslovni izrazi**

12. Napisati funkciju koja vraća znak datog broja.

- **Svako if ima svoje else.**
  - Zašto?

Ograđene jednačine

- Alternativni način definisanja uslovnih izraza;
  - Niz logičkih izraza (čuvara) na osnovu kojih se bira rezultat (svi ponuđeni rezultati su istog tipa);

```
abs n | n ≥ 0     = n
      | otherwise = -n
```

  - otherwise = *True*

Zadaci:

13. Napisati funkciju znak pomoću ograđenih jednačina.

Poklapanje šablona

- Na osnovu niza obrazaca bira se odgovarajući rezultat (svi ponuđeni rezultati su istog tipa).

```
not       :: Bool → Bool
not False = True
not True  = False
```

not slika False u True, True u False.

Poklapanje šablona

- Operator konjukcije

```
(&&)             :: Bool → Bool → Bool
True  && True  = True
True  && False = False
False && True  = False
False && False = False
```

može biti definisan na kompaktniji način:

```
True && True = True
_    && _    = False
```

Poklapanje šablona

- Efikasnija definicija

```
True  && b = b

False && _ = False
```

Poklapanje šablona

- Poklapanje šablona se provjerava redom.

```
_    && _    = False
True && True = True
```

- Nije dozvoljeno ponavljanje imena u šablonima


```
b && b = b
_ && _ = False
```

$$b \wedge c \mid b == c \quad = \quad b$$
$$\mid otherwise \quad = \quad False$$

Zadaci:

14. Definisati disjunkciju tehnikom poklapanja šablona.

Poklapanje šablona - torke

- Torka je sama po sebi šablon.
  - Kad se poklapaju dvije torke?

$$
\begin{array}{lcl}
fst & :: & (a, b) \rightarrow a \\
fst\ (x, \_) & = & x \\
snd & :: & (a, b) \rightarrow b \\
snd\ (\_, y) & = & y
\end{array}
$$

Poklapanje šablona - liste

- Lista je sama po sebi šablon.
  - Kad se poklapaju dvije liste?

Šta je svrha funkcije test?

$$
\begin{array}{lcl}
test & :: & [Char] \rightarrow Bool \\
test\ [\texttt{'a'}, \_, \_] & = & True \\
test\ \_ & = & False
\end{array}
$$

- Kako se formiraju liste?

$$
\begin{array}{ll}
 & [1,2,3] \\
= & \quad \{ \text{ list notation } \} \\
 & 1 : [2,3] \\
= & \quad \{ \text{ list notation } \} \\
 & 1 : (2 : [3]) \\
= & \quad \{ \text{ list notation } \} \\
 & 1 : (2 : (3 : [\,])) \\
\end{array}
$$

Cons operator : je desno asocijativan!

Poklapanje šablona - liste

- $x{:}xs$ – šablon neprazne liste;
- Opštija definicija funkcije test:

```
> head []
ERROR
```

$$
\begin{array}{lcl}
test & :: & [Char] \rightarrow Bool \\
test\ (\texttt{'a'} : \_) & = & True \\
test\ \_ & = & False
\end{array}
$$

- Šablon $(x{:}xs)$ se navodi u zagradama jer primjena funkcija ima veći prioritet od cons operatora.


```
head x:_ = x
```

Zadaci:

15. Tehnikom uparivanja šablona napisati funkciju koja provjerava da li je data lista prazna.
16. Tehnikom uparivanja šablona napisati funkciju koja vraća glavu liste.
17. Tehnikom uparivanja šablona napisati funkciju koja vraća rep liste.

Poklapanje šablona – cijeli brojevi

Prolazi u hugs,
ali ne i u ghci.

- Šablon $n + k$, gdje je n cijelobrojna varijabla, a $k > 0$ cjelobrojna konstanta.

$$
\begin{array}{lcl}
pred & :: & Int \rightarrow Int \\
pred\ 0 & = & 0 \\
pred\ (n+1) & = & n
\end{array}
$$

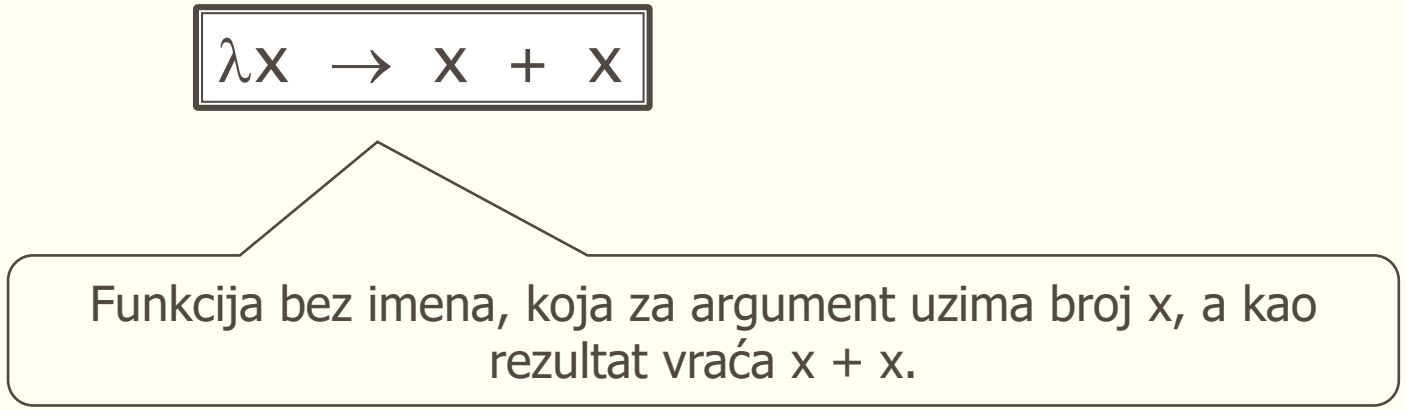
- Sa kojim cijelim brojevima se poklapa ovaj šablon?
- $pred(-1) = ?$
- $pred\ n + 1 = n\ je\ (pred\ n) + 1 = n$

LAMBDA IZRAZI

Definisanje funkcija

Lambda izrazi

- Šta su lambda funkcije?



- U ghci:

```
PS C:\Users\HP\Desktop> GHCI
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> (\ x-> x+x) 2
4
```

Lambda izrazi

- Formalizovanje definisanja Karijevih funkcija.

```
add x y = x + y
```

```
add = λx → (λy → x + y)
```

- Definisanje funkcija koje vraćaju funkciju kao rezultat.

```
const    :: a → b → a
const x _ = x
```

```
const  :: a → (b → a)
const x = λ_ → x
```

Lambda izrazi

- Izbjegavanje imenovanja funkcija koje se koriste samo jednom.

```
odds n = map f [0..n-1]
         where
             f x = x*2 + 1
```


```
odds n = map (λx → x*2 + 1) [0..n-1]
```

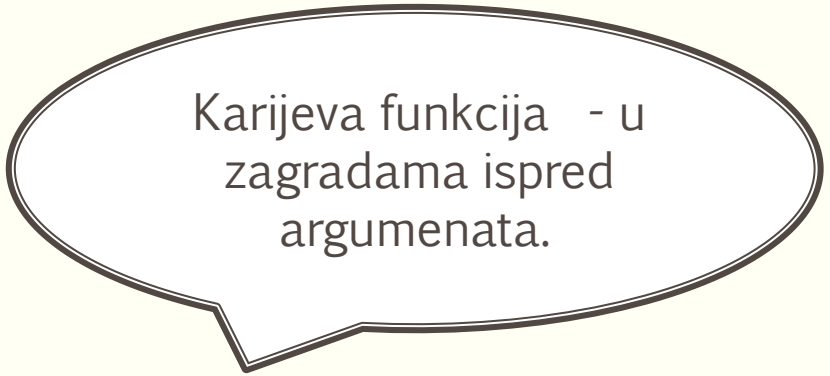
SEKCIJE

Definisanje funkcija

Sekcije

- Funkcija → operator?
  - $7\ `div`\ 2$
- Operator → funkcija?

```
> 1+2
3

> (+) 1 2
3
```



Sekcije

- Po konvenciji može i neki od argumenata da se nađe unutar zagrada.

```
> (1+) 2
3

> (+2) 1
3
```

Sekcije

- U opštem slučaju, ako je $\oplus$ operator onda su funkcije oblika $(\oplus)$, $(x\oplus)$ i $(\oplus y)$ sekcije.

$$
\begin{aligned}
(\oplus) &= \lambda x \to (\lambda y \to x \oplus y) \\
(x \oplus) &= \lambda y \to x \oplus y \\
(\oplus y) &= \lambda x \to x \oplus y
\end{aligned}
$$

Sekcije

- Definisanje jednostavnih, ali korisnih funkcija na kompaktan način.

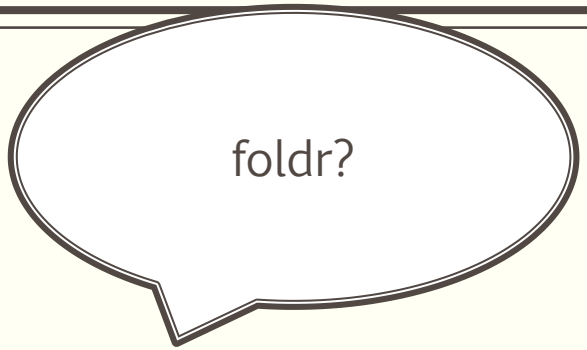
$$
\begin{aligned}
(+) &\quad \text{is the addition function } \lambda x \to (\lambda y \to x + y) \\
(1+) &\quad \text{is the successor function } \lambda y \to 1 + y \\
(1/) &\quad \text{is the reciprocation function } \lambda y \to 1 \,/\, y \\
(*2) &\quad \text{is the doubling function } \lambda x \to x * 2 \\
(/2) &\quad \text{is the halving function } \lambda x \to x \,/\, 2
\end{aligned}
$$

- Tipiziranje operatora;

$$(\wedge) \quad :: \quad Bool \to Bool \to Bool$$

Sekcije

- Operatori kao argumenti drugih funkcija.



$$
\begin{array}{lcl}
and & :: & [Bool] \rightarrow Bool \\
and & = & foldr\ (\wedge)\ True
\end{array}
$$

Zadaci:

18. Koristeći ugrađene funkcije, definisati funkciju

$$halve :: [a] \rightarrow ([a], [a])$$

koja listu parne dužine dijeli na dvije polovine. Npr.

$$\begin{array}{l} > \ halve\ [1,2,3,4,5,6] \\ ([1,2,3],[4,5,6]) \end{array}$$

19. Napisati funkciju

$$safetail :: [a] \rightarrow [a]$$

koja se ponaša isto kao i funkcija $tail$, a za praznu listu vraća praznu listu. Funkciju definisati:

a. Uslovnim izrazima;
b. Ograđenim jednačinama;
c. Poklapanjem šablona.

Zadaci:

20. Definisati operator konjukcije dat sljedećim upotrebom uslovnih izraza.

$$\begin{aligned} True \wedge True &= True \\ \_ \wedge \_ &= False \end{aligned}$$

21. Definisati operator konjukcije dat sljedećim upotrebom uslovnih izraza.

$$\begin{aligned} True \wedge b &= b \\ False \wedge \_ &= False \end{aligned}$$

22. Definisati sljedeću funkciju lambda izrazom

$$mult\ x\ y\ z = x * y * z$$

ZADAVANJE LISTI

Haskell

Generatori

- Matematika – zadavanje skupa na osnovu postojećeg skupa

$$\{x^2 \mid x \in \{1...5\}\}$$

- Haskell – zadavanje liste na osnovu postojeće liste

```
[x^2 | x ← [1..5]]
```

- | - „takvi da“; ← „iz“; izraz x ← [1..5] - generator;

Generatori

- Više uzastopnih generatora odvojenih zarezima

```
> [(x,y) | x ← [1,2,3], y ← [4,5]]

[(1,4),(1,5),(2,4),(2,5),(3,4),(3,5)]
```

„kasniji“
vs „raniji“
generatori

- Promjena redoslijeda generatora

```
> [(x,y) | y ← [4,5], x ← [1,2,3]]

[(1,4),(2,4),(3,4),(1,5),(2,5),(3,5)]
```

Generatori

- Međusobna zavisnost generatora

```
[(x,y) | x ← [1..3], y ← [x..3]]
```

Generatori

- Međusobna zavisnost generatora

```
[(x,y) | x ← [1..3], y ← [x..3]]
```

Lista [(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)] svih parova brojeva (x,y) takvih da su x,y elementi liste [1..3] i da je $y \geq x$.

Zadaci:

23. Definisati ugrađenu funkciju concat tehnikom zadavanja listi.
24. Tehnikom zadavanja listi iz liste uređenih parova izdvojiti listu prvih komponenti.
25. Definisati ugrađenu funkciju length tehnikom zadavanja listi.

Čuvari

- Logički izrazi za filtriranje vrijednosti generisanih generatorom.

```
[x | x ← [1..10], even x]
```

Lista [2,4,6,8,10] svih brojeva x takvih da je x element liste [1..10] i da je x paran.

Zadaci:

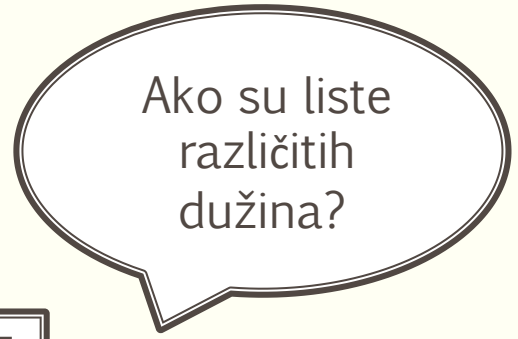
26. Tehnikom zadavanja listi napisati funkciju koja za argument uzima cijeli broj, a kao rezultat vraća listu djelilaca datog broja.
27. Tehnikom zadavanja listi napisati funkciju koja ispituje da li je cijeli broj, koji uzima za argument, prost broj.
28. Tehnikom zadavanja listi napisati funkciju koja vraća listu svih prostih brojeva manjih od cijelog broja koji je argument funkcije.
29. Neka je data lista uređenih parova (key, value). Tehnikom zadavanja listi napisati funkciju koja vraća listu svih vrijednosti koje su pridružene datom ključu.

Zip funkcija

- Funkcija koja na osnovu dvije liste formira listu uređenih parova odgovarajućih elemenata.

```
zip :: [a] → [b] → [(a,b)]
```

Na primjer:



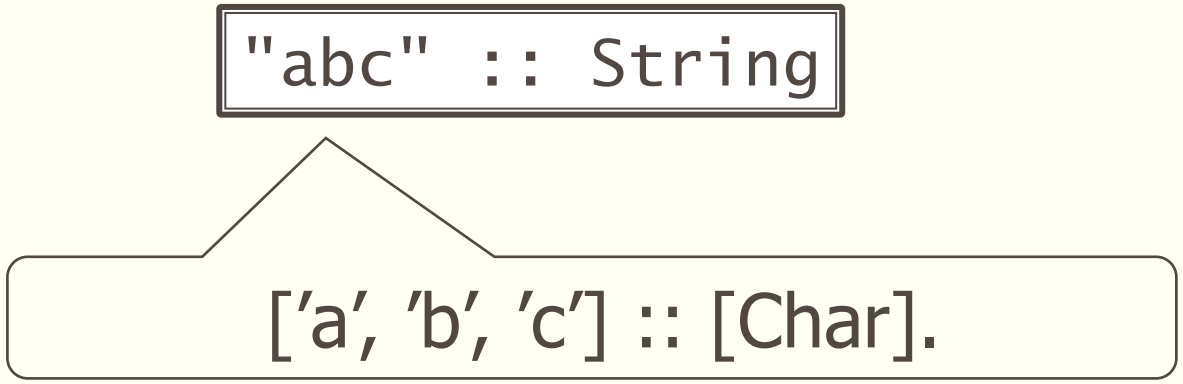

```
> zip ['a','b','c'] [1,2,3,4]

[('a',1),('b',2),('c',3)]
```

Zadaci:

30. Napisati funkciju koja za argument uzima listu elemenata, a kao rezultat vraća listu parova uzastopnih elementa te liste.
31. Napisati funkciju koja za argument uzima listu i ispituje da li su elementi liste sortirani u neopadajućem poretku.
32. Napisati funkciju koja za argument uzima listu i element, a kao rezultat vraća listu svih pozicija na kojima se dati element pojavljuje u datoj listi.

Stringovi

- String je niz karaktera unutar dvostrukih navodnika.
- String je lista karaktera.



Stringovi

- Polimorfne funkcije definisane na listama, mogu da se koriste na stringovima.

```
> length "abcde"
5

>"abcde"!!2
‘c’

> take 3 "abcde"
"abc"

> zip "abc" [1,2,3,4]
[(’a’,1),(’b’,2),(’c’,3)]
```

Zadaci:

33. Tehnikom zadavanja listi definisati funkciju koja za argument uzima string i vraća informaciju koliko malih slova se nalazi u tom stringu.

34. Tehnikom zadavanja listi definisati funkciju koja za argument uzima string i karakter, a vraća informaciju koliko puta se dati karakter pojavljuje u datom stringu.

Zadaci za vježbu

- Cezarov disk;
- 5.7. (strane 60-61);

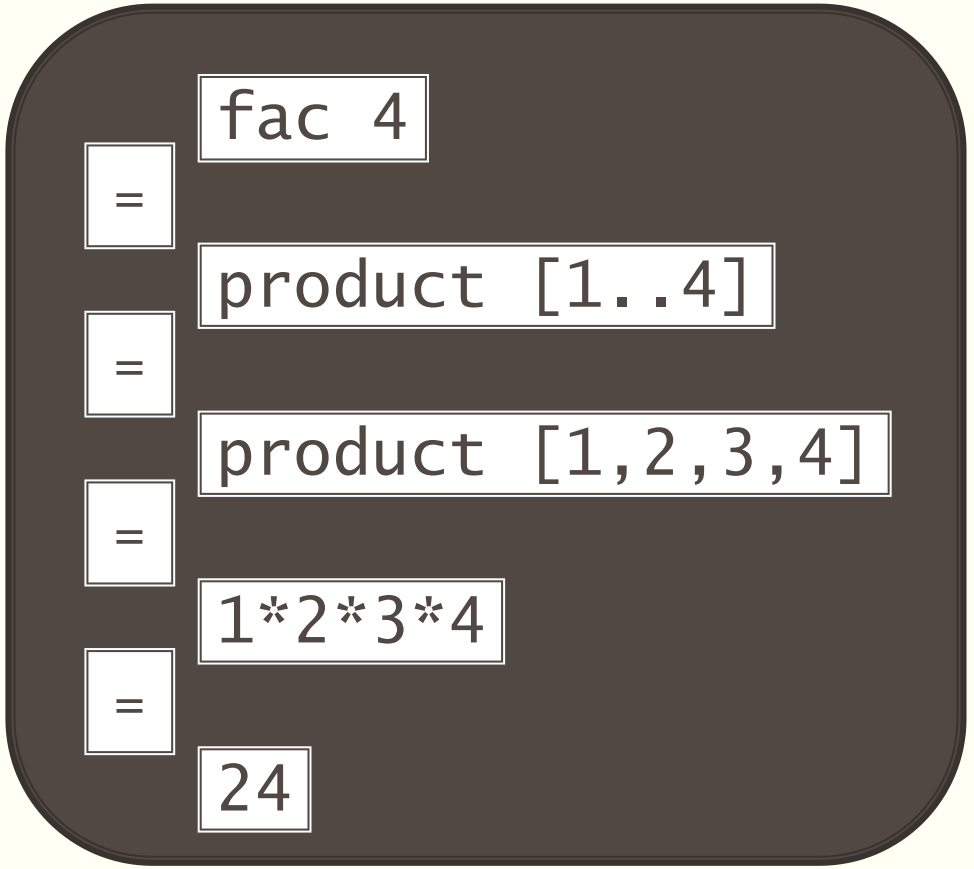
REKURZIVNE FUNKCIJE

Haskell

Osnovni koncepti

- Definisanje novih funkcija na osnovu postojećih funkcija.

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



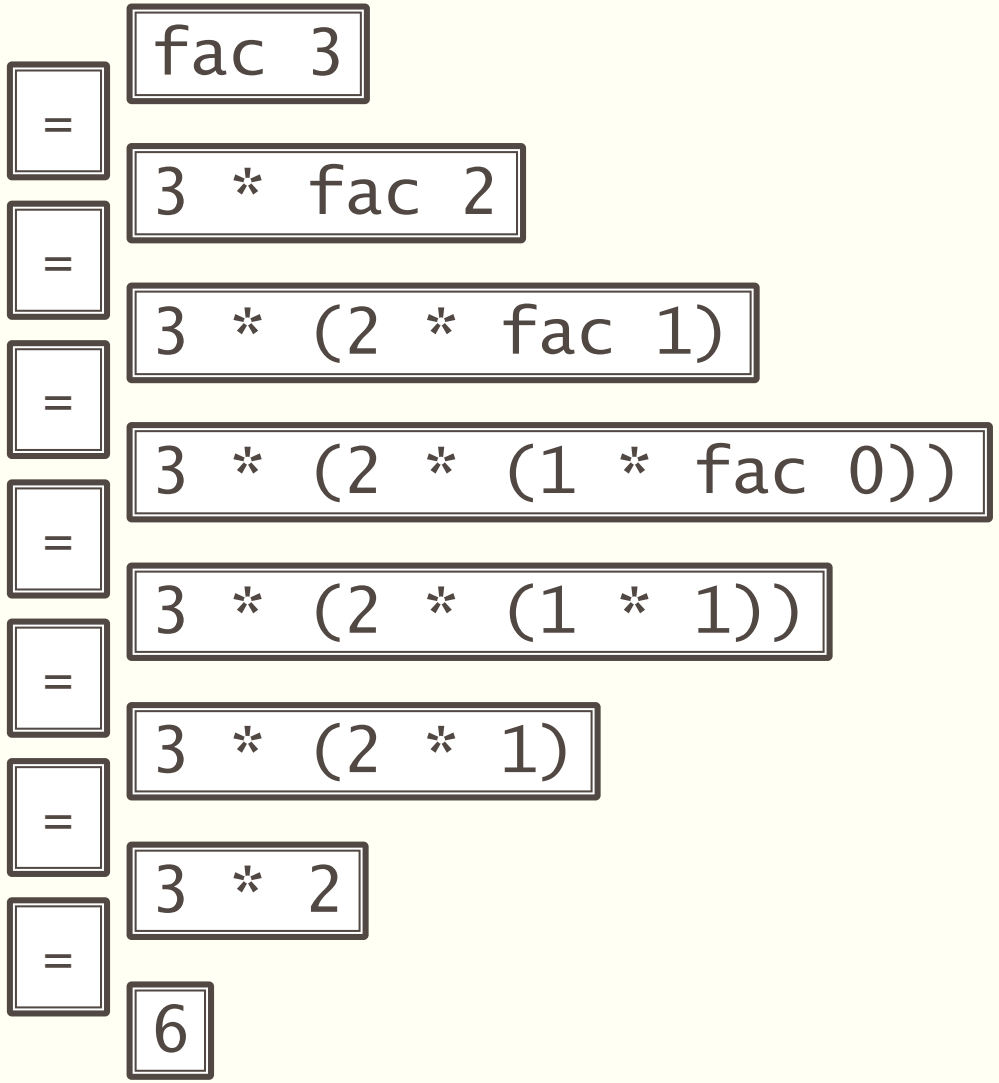
Osnovni koncepti

- Definisanje funkcije koja poziva sama sebe.

Bazni slučaj;
Rekurzivni slučaj;

```
fac 0 = 1
fac n = n * fac (n-1)
```

```
> fac (-1)

Exception: stack overflow
```


fac 3
=
3 * fac 2
=
3 * (2 * fac 1)
=
3 * (2 * (1 * fac 0))
=
3 * (2 * (1 * 1))
=
3 * (2 * 1)
=
3 * 2
=
6

Prednosti upotrebe rekurzije

- Jednostavnije definicije nekih funkcija.
- Neke funkcije se prirodnije definišu na rekurzivan način.
- Osobine rekurzivnih funkcija se mogu pokazati matematičkom indukcijom.

Osnovni koncepti

- Rekurzivna definicija operatora * za nengativne cijele brojeve

$$
\begin{array}{lcl}
(*) & :: & Int \to Int \to Int \\
m * 0 & = & 0 \\
m * (n+1) & = & m + (m * n)
\end{array}
$$

$$
\begin{array}{ll}
 & 4 * 3 \\
= & \quad \{ \text{ applying } * \} \\
 & 4 + (4 * 2) \\
= & \quad \{ \text{ applying } * \} \\
 & 4 + (4 + (4 * 1)) \\
= & \quad \{ \text{ applying } * \} \\
 & 4 + (4 + (4 + (4 * 0))) \\
= & \quad \{ \text{ applying } * \} \\
 & 4 + (4 + (4 + 0)) \\
= & \quad \{ \text{ applying } + \} \\
 & 12
\end{array}
$$

Rekurzija nad listama

```
product        :: Num a ⇒ [a] → a
product []      = 1
product (n:ns) = n * product ns
```

Rekurzija nad listama

$$
\begin{aligned}
& \mathit{product}\ [2,3,4] \\
= & \qquad \{\ \text{applying } \mathit{product}\ \} \\
& 2 * \mathit{product}\ [3,4] \\
= & \qquad \{\ \text{applying } \mathit{product}\ \} \\
& 2 * (3 * \mathit{product}\ [4]) \\
= & \qquad \{\ \text{applying } \mathit{product}\ \} \\
& 2 * (3 * (4 * \mathit{product}\ [\,])) \\
= & \qquad \{\ \text{applying } \mathit{product}\ \} \\
& 2 * (3 * (4 * 1)) \\
= & \qquad \{\ \text{applying } *\ \} \\
& 24
\end{aligned}
$$

Zadaci:

35. Rekurzivno definisati funkciju length.
36. Rekurzivno definisati funkciju reverse.
37. Rekurzivno definisati ++.
38. Rekurzivno definisati insert sort.

Rekurzija sa više argumenata

- Rekurzija na više argumenata istovremeno;

Zadaci:

39. Rekurzivno definisati funkciju zip.
40. Rekurzivno definisati funkciju drop.

Primjena rekurzivno definisane funkcije zip

Zašto dva bazna slučaja?

$$
\begin{aligned}
& zip\ [\texttt{'a'},\texttt{'b'},\texttt{'c'}]\ [1,2,3,4] \\
= & \quad \{\ \text{applying } zip\ \} \\
& (\texttt{'a'},1) : zip\ [\texttt{'b'},\texttt{'c'}]\ [2,3,4] \\
= & \quad \{\ \text{applying } zip\ \} \\
& (\texttt{'a'},1) : (\texttt{'b'},2) : zip\ [\texttt{'c'}]\ [3,4] \\
= & \quad \{\ \text{applying } zip\ \} \\
& (\texttt{'a'},1) : (\texttt{'b'},2) : (\texttt{'c'},3) : zip\ []\ [4] \\
= & \quad \{\ \text{applying } zip\ \} \\
& (\texttt{'a'},1) : (\texttt{'b'},2) : (\texttt{'c'},3) : [] \\
= & \quad \{\ \text{list notation}\ \} \\
& [(\texttt{'a'},1),(\texttt{'b'},2),(\texttt{'c'},3)]
\end{aligned}
$$

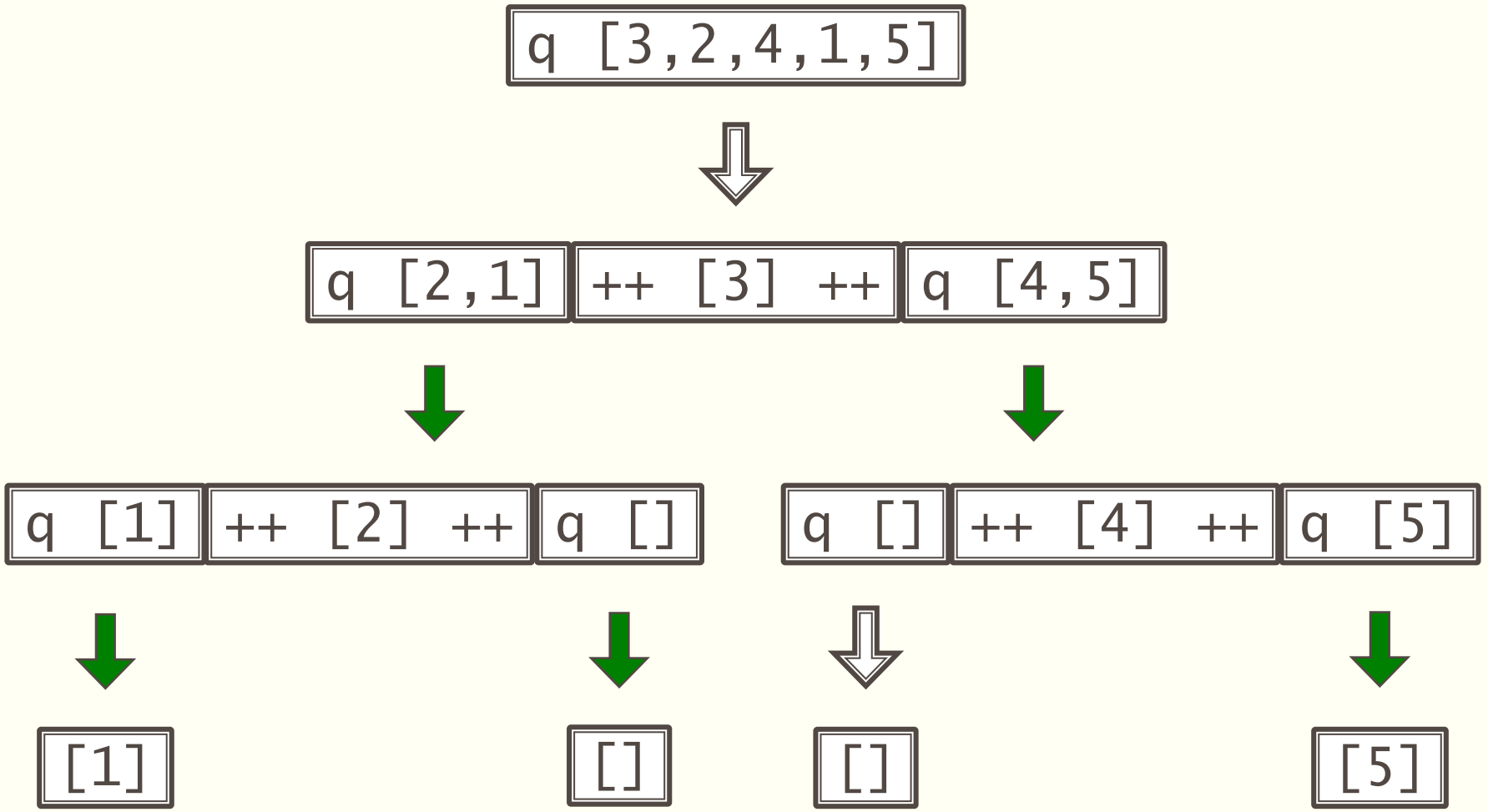
Višestruka rekurzija

- Više od jednog rekurzivnog poziva u definiciji funkcije.

Zadaci:

41. Rekurzivno definisati funkciju koja vraća n-ti Fibonačijev broj, pri čemu je broj n argument funkcije.
42. Rekurzivno definisati quick sort.

Na primjer (qsort je označen sa q):



Quicksort

- Ideja:
  - Prazna lista je sortirana.
  - Neprazna lista postaje soritirana ako se glava liste postavi između sortiranih elemenata repa liste koji su manji od nje i sortiranih elemenata repa liste koji su veći od nje.
- Implementacija quicksort-a u Haskell-u vs implementacija quick sort-a u drugim programskim jezicima (Java, C,...)?

$$
\begin{aligned}
&qsort && :: && Ord\ a \Rightarrow [a] \rightarrow [a] \\
&qsort\ [\,] && = && [\,] \\
&qsort\ (x : xs) && = && qsort\ smaller +\!\!+ [x] +\!\!+ qsort\ larger \\
& && && \textbf{where} \\
& && && \quad smaller = [a \mid a \leftarrow xs, a \leqslant x] \\
& && && \quad larger = [b \mid b \leftarrow xs, b > x]
\end{aligned}
$$

Rekurzivno definisati *smaller* i *larger*!

Međusobna rekurzija

- Dvije ili više funkcija je definisano u terminima jedna druge.

Zadaci:

43. Rekurzivno definisati funkciju koja ispituje da li je broj koji uzima za argument paran.
44. Rekurzivno definisati funkciju koja ispituje da li je broj koji uzima za argument neparan.

Rješenje 1. – rekurzivni poziv iste funkcije

```
paran::Int->Bool
paran 0 = True
paran 1 = False
paran n = paran (n-2)
```

```
neparan::Int->Bool
neparan 0 = False
neparan 1 = True
neparan n = neparan (n-2)
```

Rješenje 2. – međusobni rekurzivni pozivi

Rješenje 1 vs
Rješenje 2

```
paran::Int->Bool
paran 0 = True
paran n = neparan (n-1)
```

```
neparan::Int->Bool
neparan 0 = False
neparan n = paran (n-1)
```

Međusobna rekurzija

- Može se definisati i na listama.

$$
\begin{array}{lcl}
evens & :: & [a] \to [a] \\
evens\ [\,] & = & [\,] \\
evens\ (x : xs) & = & x : odds\ xs \\
odds & :: & [a] \to [a] \\
odds\ [\,] & = & [\,] \\
odds\ (\_ : xs) & = & evens\ xs
\end{array}
$$

$$
\begin{array}{ll}
 & evens\ \texttt{"abcde"} \\
= & \quad \{\ \text{applying } evens\ \} \\
 & \texttt{'a'} : odds\ \texttt{"bcde"} \\
= & \quad \{\ \text{applying } odds\ \} \\
 & \texttt{'a'} : evens\ \texttt{"cde"} \\
= & \quad \{\ \text{applying } evens\ \} \\
 & \texttt{'a'} : \texttt{'c'} : odds\ \texttt{"de"} \\
= & \quad \{\ \text{applying } odds\ \} \\
 & \texttt{'a'} : \texttt{'c'} : evens\ \texttt{"e"} \\
= & \quad \{\ \text{applying } evens\ \} \\
 & \texttt{'a'} : \texttt{'c'} : \texttt{'e'} : odds\ [\,] \\
= & \quad \{\ \text{applying } odds\ \} \\
 & \texttt{'a'} : \texttt{'c'} : \texttt{'e'} : [\,] \\
= & \quad \{\ \text{string notation}\ \} \\
 & \texttt{"ace"}
\end{array}
$$

Koraci pri definisanju rekurzivnih funkcija

- Korak 1. Definisati tip funkcije

$$product \quad :: \quad [Int] \rightarrow Int$$

- Korak 2. Nabrojati standardne slučajeve.

$$\begin{array}{ll} product\ [\,] & = \\ product\ (n : ns) & = \end{array}$$

Koraci pri definisanju rekurzivnih funkcija

- Korak 3. Definisati jednostavne slučajeve.

$$\begin{array}{lcl} product\ [\,] & = & 1 \\ product\ (n : ns) & = & \end{array}$$

- Korak 4. Definisati ostale slučajeve.

$$\begin{array}{lcl} product\ [\,] & = & 1 \\ product\ (n : ns) & = & n * product\ ns \end{array}$$

Koraci pri definisanju rekurzivnih funkcija

- Korak 5. Uopštiti i pojednostaviti.

$$product \quad :: \quad Num\ a \Rightarrow [a] \rightarrow a$$

$$product \quad = \quad foldr\ (*)\ 1$$

- Konačno

$$\begin{aligned} product \quad &:: \quad Num\ a \Rightarrow [a] \rightarrow a \\ product \quad &= \quad foldr\ (*)\ 1 \end{aligned}$$

Zadaci:

45. Rekurzivno definisati funkciju koja računa n-ti stepen cijelog broja.
46. Rekurzivno definisati funkciju koja računa maksimalan broj presjeka n pravih u ravni.
47. Rekurzivno definisati funkciju koja računa sumu kvadrata prvih n prirodnih brojeva.
48. Rekurzivno definisati funkciju koja računa najmanji djelilac broja n.
49. Rekurzivno definisati funkciju koja za argument uzima paran broj n i računa sumu svih parnih brojeva od 0 do $n/2$.
50. Rekurzivno definisati funkciju and.
51. Rekurzivno definisati funkciju replicate.
52. Rekurzivno definisati funkciju !!.
53. Rekurzivno definisati funkciju elem.

Zadaci za vježbu:

- Primjeri funkcija drop i init (strane 72-75);
- Zadaci 6.8 (strana 75);

FUNKCIJE VIŠEG REDA

Haskell

Osnovni koncepti

- Funkcije sa više argumenata su definisane u notaciji Karijevih funkcija.

$$
\begin{array}{lcl}
add & :: & Int \rightarrow Int \rightarrow Int \\
add\ x\ y & = & x + y
\end{array}
$$

$$
\begin{array}{lcl}
add & :: & Int \rightarrow (Int \rightarrow Int) \\
add & = & \lambda x \rightarrow (\lambda y \rightarrow x + y)
\end{array}
$$

Osnovni koncepti

- U Haskell-u je moguće definisati funkcije koje za argument uzimaju funkcije.

$$
\begin{array}{lcl}
twice & :: & (a \to a) \to a \to a \\
twice\ f\ x & = & f\ (f\ x)
\end{array}
$$

(*2)?

$$
\begin{array}{l}
> \ twice\ (*2)\ 3 \\
12
\end{array}
$$

Koja je ovo funkcija?

$$
\begin{array}{l}
> \ twice\ reverse\ [1,2,3] \\
[1,2,3]
\end{array}
$$

Parcijalna primjena -> nove korisne funkcije!

Osnovni koncepti

- Funkcija koja za argument uzima funkciju ili vraća funkciju kao rezultat je **funkcija višeg reda.**
  - Karijeve funkcije?
- Zašto su korisne funkcije višeg reda?
  - Česti obrasci se inkapsuliraju u funkcije.
  - Domenski specifični jezici se mogu definisati kao kolekcije funkcija višeg reda.

Procesiranje listi

- Ugrađene funkcije višeg reda za rad sa listama.
- *map*


```
> map (+1) [1,3,5,7]
[2,4,6,8]

> map isDigit ['a','1','b','2']
[False,True,False,True]

> map reverse ["abc","def","ghi"]
["cba","fed","ihg"]
```

map funkcija

- map je funkcija višeg reda koja ima dva argumenta, funkciju f i listu l, a kao rezultat vraća listu koja sadrži rezultat primjene funkcije f na elemente liste l.
- map funkcija:
  - Polimorfna funkcija;
  - Može biti primjenja „sama na sebe“ da bi se postiglo procesiranje ugnježdenih listi;

$$map\ (map\ (+1))$$

$$\begin{array}{ll} & map\ (map\ (+1))\ [[1,2,3],[4,5]] \\ = & \quad \{\ \text{applying the outer } map\ \} \\ & [map\ (+1)\ [1,2,3], map\ (+1)\ [4,5]] \\ = & \quad \{\ \text{applying } map\ \} \\ & [[2,3,4],[5,6]] \end{array}$$

Zadaci:

54. Definisati funkciju map:
    - tehnikom zadavanja listi;
    - rekurzivno.

Procesiranje listi

- *filter* funkcija

Tip funkcije *filter*?

```
> filter even [1..10]
[2,4,6,8,10]

> filter (>5) [1..10]
[6,7,8,9,10]

> filter (≠ '␣') "abc␣def␣ghi"
"abcdefghi"
```

$filter$ funkcija

- $filter$ funkcija za argument uzima predikat p i listu l, a kao rezultat vraća listu elemenata liste l koji zadovoljavaju dati predikat p.
  - Za funkciju kažemo da je predikat ako je rezultat funkcije podatak tipa *Bool*, dakle *True* ili *False*.

Zadaci:

55. Definisati funkciju $filter$:
    - tehnikom zadavanja listi;
    - rekurzivno.
56. Upotrebom funkcija map i $filter$, kao i ugrađenih funkcija, napisati funkciju koja sumira kvadrate parnih brojeva date liste.
57. Napisati rekurzivnu definiciju funkcije iz zadatka 3.

Još neke ugrađene funkcije višeg reda za rad sa listama

- *all* funkcija za argumente uzima predikat p i listu l, a kao rezultat vraća informaciju da li svi elementi liste l zadovoljavaju dati predikat p.

```
> all even [2, 4, 6, 8]
True
```

- *any* funkcija za argumente uzima predikat p i listu l, a kao rezultat vraća informaciju da li bar jedan elementi liste l zadovoljava dati predikat p.

```
> any odd [2, 4, 6, 8]
False
```

Još neke ugrađene funkcije višeg reda za rad sa listama

- *takeWhile* funkcija za argumente uzima predikat p i listu l i u rezultujuću listu izdvaja elemente liste l sve dok vrijedi predikat p.

```
> takeWhile isLower "abc def"
"abc"
```

- *dropWhile* funkcija za argumente uzima predikat p i listu l i iz date liste l izbacuje elemente sve dok vrijedi predikat p.

```
> dropWhile isLower "abc def"
" def"
```

Kog tipa su ove funkcije?

Zadaci:

58. Definisati funkciju all:
    - tehnikom zadavanja listi i ugrađenim funkcijama;
    - rekurzivno.
59. Nije dozvoljeno „miješanje tehnika“.
60. Definisati funkciju any:
    - tehnikom zadavanja listi i ugrađenim funkcijama;
    - rekurzivno.
61. Napisati rekurzivnu definiciju funkcije $takeWhile$.
62. Napisati rekurzivnu definiciju funkcije $dropWhile$.

Zadaci:

63. Upotrebom funkcija map i $filter$ napisati funkciju koja za argumene uzima funkciju f, predikat p i listu xs, a kao rezultat vraća listu koja je jednaka listi datoj sa
    - $[f\ x \mid x \leftarrow xs,\ p\ x]$.
63. Uraditi 3. zadatak tehnikom zadavanje listi i upotrebom ugrađenih funkcija.
    - Napomena: nije dozvoljena upotreba funkcija map i filter, kao ni miješanje tehnika.
64. Napisati funkciju koja ispituje da li su elementi liste soritrani rastuće. Funkciju definisati na tri načina:
    - rekurzivno,
    - tehnikom zadavanja listi i ugrađenim funkcijama.
    - funkcijom $filter$ i ugrađenim funkcijama

    Napomena: nije dozvoljena upotreba funkcija map, kao ni miješanje tehnika.

$foldr$ funkcija

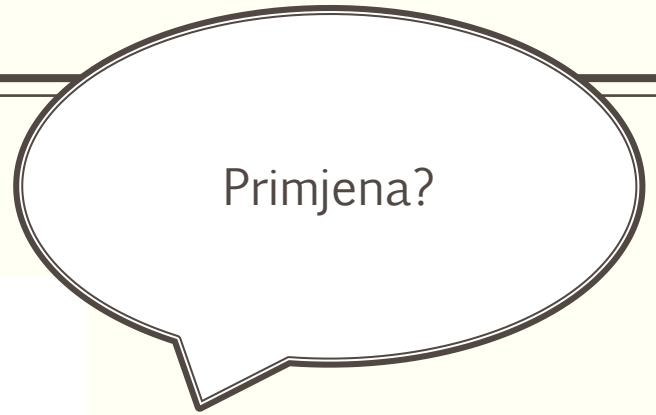
- Obrazac rekurzije na listama:

$$
\begin{array}{lcl}
f\ [\,] & = & v \\
f\ (x : xs) & = & x \oplus f\ xs
\end{array}
$$

$$
\begin{array}{lcl}
sum\ [\,] & = & 0 \\
sum\ (x : xs) & = & x + sum\ xs \\
product\ [\,] & = & 1 \\
product\ (x : xs) & = & x * product\ xs \\
or\ [\,] & = & False \\
or\ (x : xs) & = & x \vee or\ xs \\
and\ [\,] & = & True \\
and\ (x : xs) & = & x \wedge and\ xs
\end{array}
$$

$foldr$ funkcija

- $fold\ right$ – enkapsulira navedeni obrazac;



$$
\begin{aligned}
sum &= foldr\ (+)\ 0 \\
product &= foldr\ (*)\ 1 \\
or &= foldr\ (\vee)\ False \\
and &= foldr\ (\wedge)\ True
\end{aligned}
$$

$foldr$ funkcija

Šta su v i f za funkcije definisane na prethodnom slajdu?

- Rekurzivna definicija funkcije $foldr$:

$$
\begin{array}{lcl}
foldr & :: & (a \to b \to b) \to b \to [a] \to b \\
foldr\ f\ v\ [\,] & = & v \\
foldr\ f\ v\ (x : xs) & = & f\ x\ (foldr\ f\ v\ xs)
\end{array}
$$

- Nerekurzivno $foldr$: $foldr\ (+)\ 0$

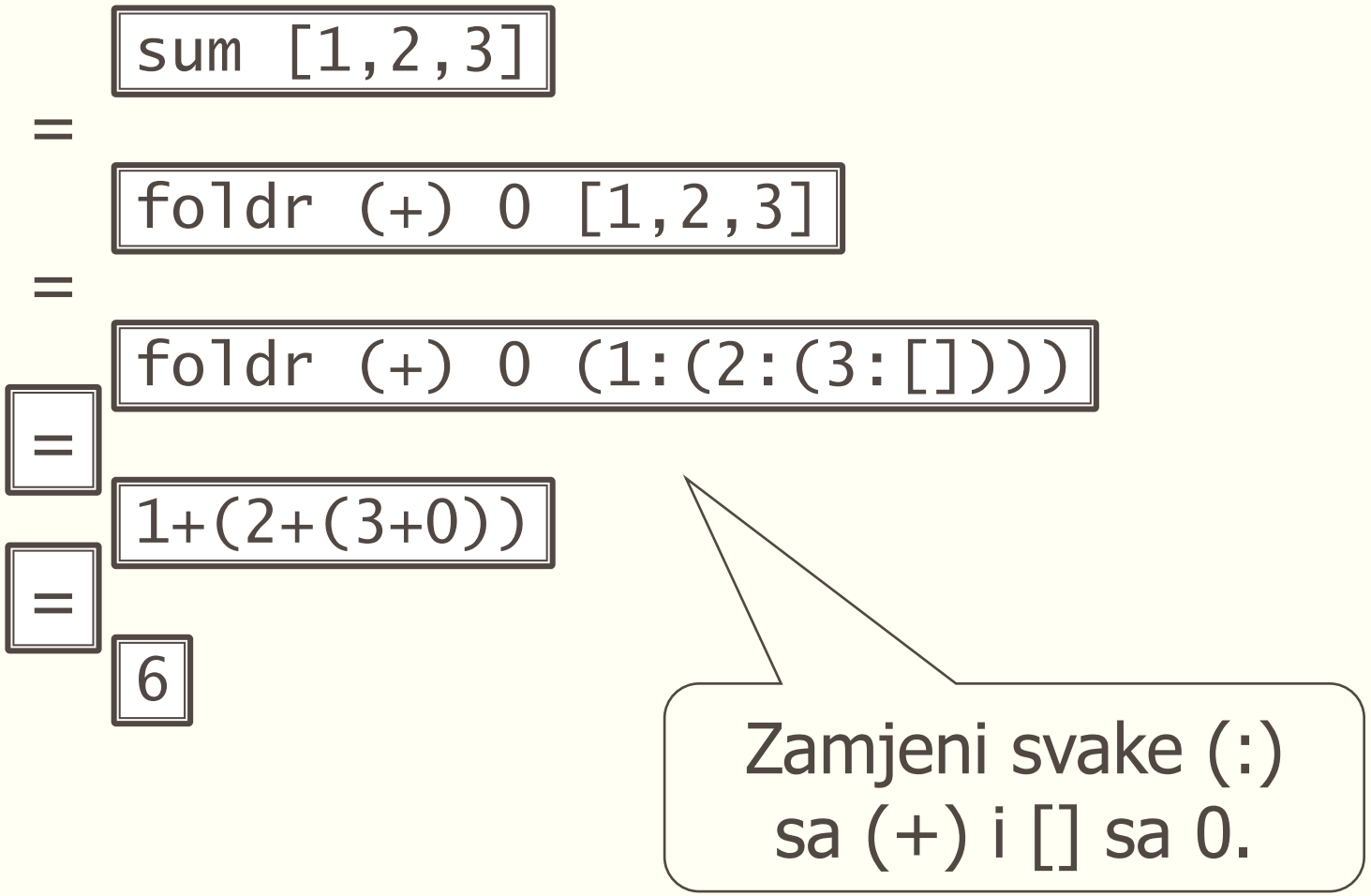
Primjena $foldr\ (+)\ 0$

$$1 : (2 : (3 : [\,]))$$

$$1 + (2 + (3 + 0))$$

Operator je desno asocijativan!

Na primjer:



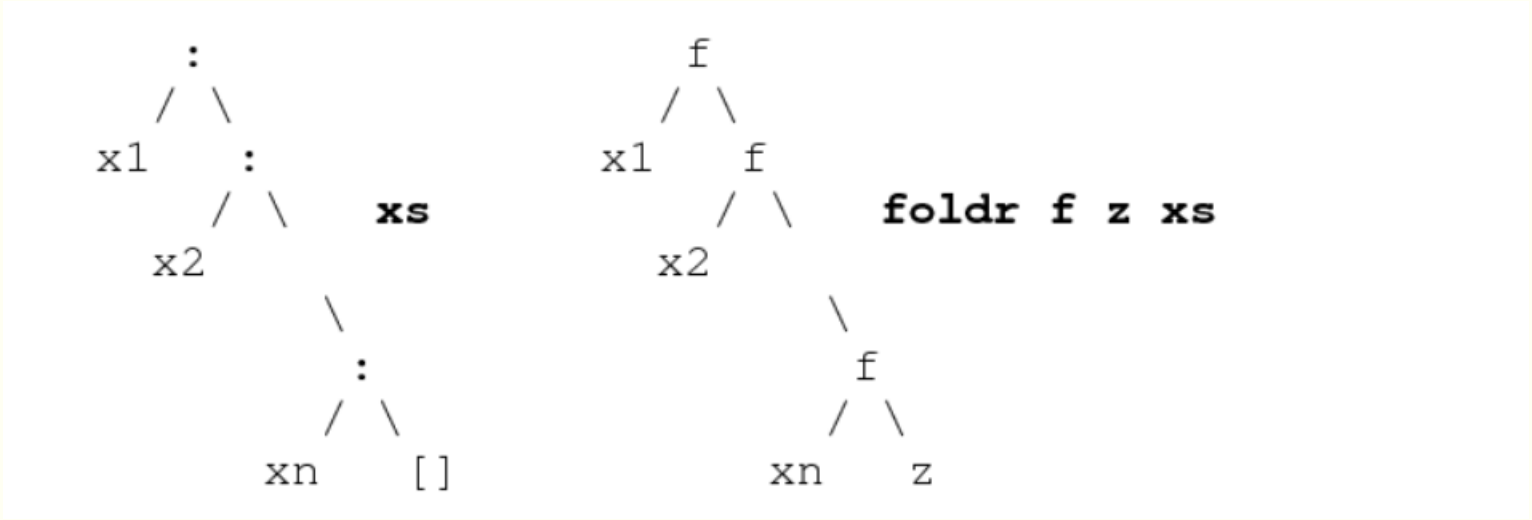
Na primjer:

```
  product [1,2,3]
=
  foldr (*) 1 [1,2,3]
=
  foldr (*) 1 (1:(2:(3:[])))
=
  1*(2*(3*1))
=
  6
```

Zamijeni svake (:) sa (*) i [] sa 1.

foldr funkcija

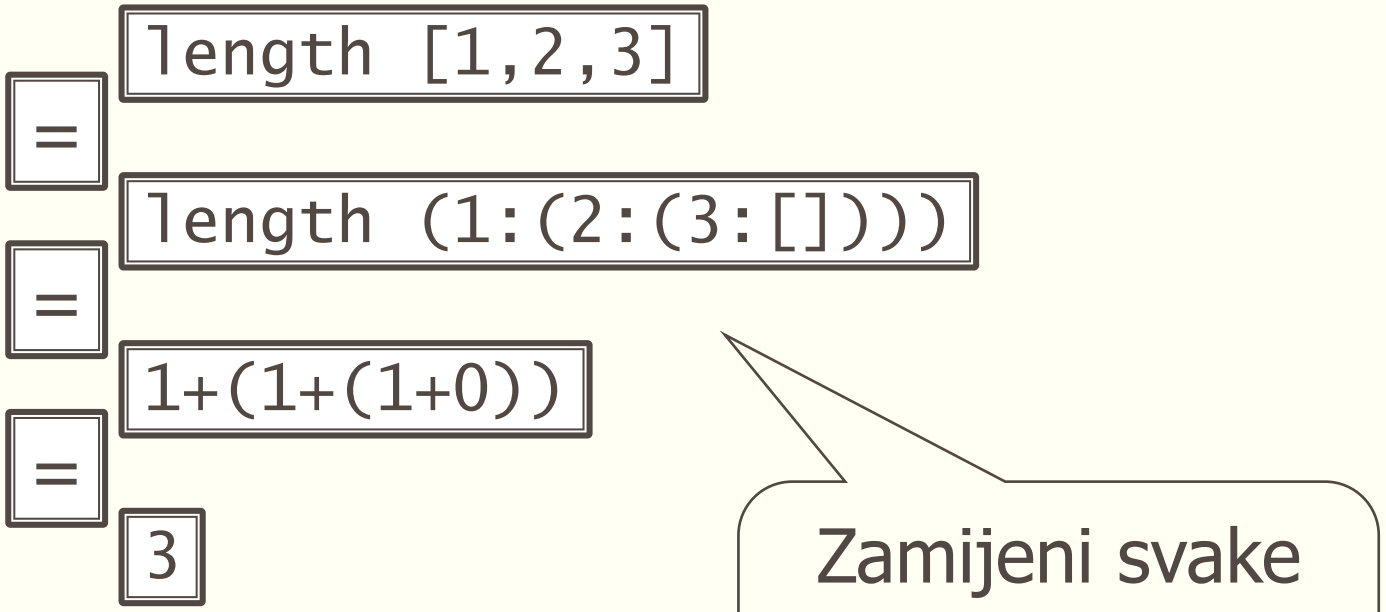
$$foldr\ (\oplus)\ v\ [x_0, x_1, \cdots, x_n] \quad = \quad x_0 \oplus (x_1 \oplus (\cdots (x_n \oplus v) \cdots))$$



Zadaci:

65. Upotrebom funkcije $foldr$ definisati funkciju $length$.
66. Upotrebom funkcije $foldr$ definisati funkciju $reverse$.
67. Definisati funkciju maksimum koja pronalazi najveći element liste pomoću funkcije $foldr$. Od pomoći može biti ugrađena funkcija max, koja vraća veći od dva data argumenta.
68. Definisati funkciju ++, koja spaja dvije liste u jednu, pomoću funkcije $foldr$.

Na primjer:



Dakle, imamo:

```
length = foldr (λ_ n → 1+n) 0
```

Funkcija reverse:

```
reverse []     = []
reverse (x:xs) = reverse xs ++ [x]
```

For example:

```
reverse [1,2,3]
=
reverse (1:(2:(3:[])))
=
(([] ++ [3]) ++ [2]) ++ [1]
=
[3,2,1]
```

Zamijeni (:) sa λx xs $\rightarrow$ xs ++ [x] i [] sa [].

Dakle, imamo:

```
reverse =
   foldr (λx xs → xs ++ [x]) []
```

Operator (++) ima kompaktnu definciju upotrebom foldr funkcije:

```
(++ ys) = foldr (:) ys
```

Zamijeni (:) sa (:) i [] sa ys.

foldl funkcija

vs sum preko foldr?

$$
\begin{aligned}
sum &= sum'\ 0 \\
&\quad \textbf{where} \\
&\qquad sum'\ v\ [\,] = v \\
&\qquad sum'\ v\ (x : xs) = sum'\ (v + x)\ xs
\end{aligned}
$$

$$
\begin{aligned}
& sum\ [1,2,3] \\
= & \quad \{\ \text{applying } sum\ \} \\
& sum'\ 0\ [1,2,3] \\
= & \quad \{\ \text{applying } sum'\ \} \\
& sum'\ (0+1)\ [2,3] \\
= & \quad \{\ \text{applying } sum'\ \} \\
& sum'\ ((0+1)+2)\ [3] \\
= & \quad \{\ \text{applying } sum'\ \} \\
& sum'\ (((0+1)+2)+3)\ [\,] \\
= & \quad \{\ \text{applying } sum'\ \} \\
& ((0+1)+2)+3 \\
= & \quad \{\ \text{applying } +\ \} \\
& 6
\end{aligned}
$$

foldl funkcija

- Obrazac rekurzije na listama:

$$\begin{aligned} f\ v\ [\,] &= v \\ f\ v\ (x : xs) &= f\ (v \oplus x)\ xs \end{aligned}$$

- Primjeri:

$$\begin{aligned} sum &= foldl\ (+)\ 0 \\ \\ product &= foldl\ (*)\ 1 \\ or &= foldl\ (\vee)\ False \\ and &= foldl\ (\wedge)\ True \end{aligned}$$

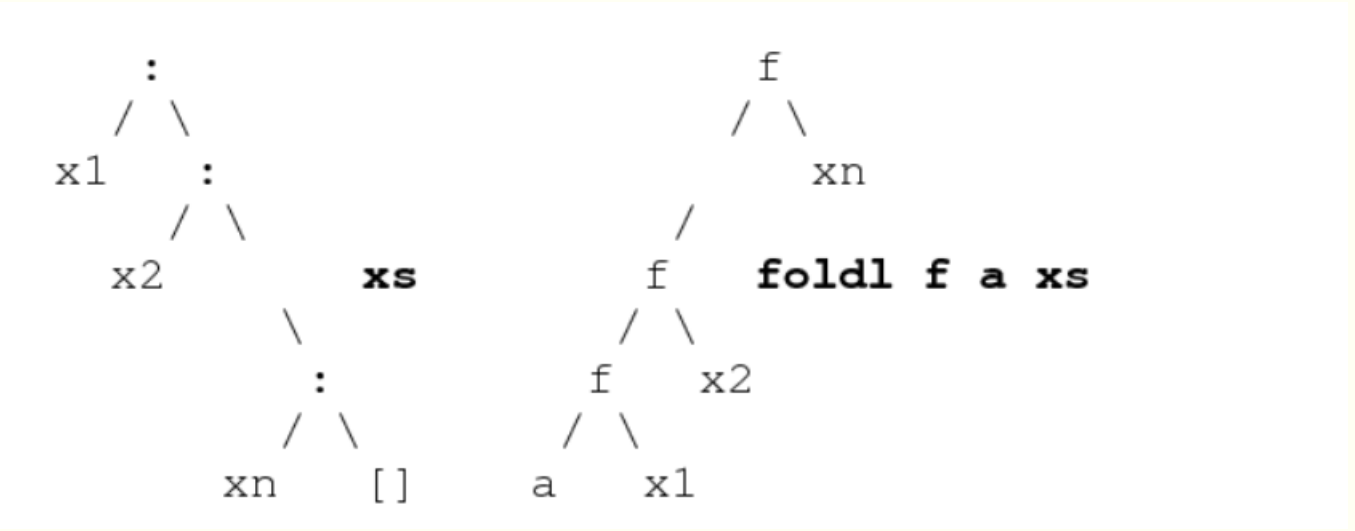
$foldl$ funkcija

- Rekurzivna definicija funkcije $foldl$:

$$
\begin{array}{lcl}
foldl & :: & (a \to b \to a) \to a \to [b] \to a \\
foldl\ f\ v\ [\,] & = & v \\
foldl\ f\ v\ (x : xs) & = & foldl\ f\ (f\ v\ x)\ xs
\end{array}
$$

foldl funkcija

$$foldl\ (\oplus)\ v\ [x_0, x_1, \cdots, x_n] \quad = \quad (\cdots\ ((v\ \oplus\ x_0)\ \oplus\ x_1)\ \cdots)\ \oplus\ x_n$$



foldr vs *foldl*

$$\mathit{foldr}\ (\oplus)\ v\ [x_0, x_1, \cdots, x_n] \quad = \quad x_0 \oplus (x_1 \oplus (\cdots (x_n \oplus v) \cdots))$$

$$\mathit{foldl}\ (\oplus)\ v\ [x_0, x_1, \cdots, x_n] \quad = \quad (\cdots ((v \oplus x_0) \oplus x_1) \cdots) \oplus x_n$$

Zadaci:

Uporediti sa defincijama pomoću foldr!

69. Upotrebom funkcije $foldl$ definisati funkciju $length$.
70. Upotrebom funkcije $foldl$ definisati funkciju $reverse$.
71. Definisati funkciju ++, koja spaja dvije liste u jednu, pomoću funkcije $foldl$.

foldl funkcija

- Primjena prethodno definisanih funkcija:

$$
\begin{aligned}
\mathit{length}\ [1,2,3] &= ((0+1)+1)+1 \\
\mathit{reverse}\ [1,2,3] &= 3:(2:(1:[])) \\
[1,2,3] +\!\!+ [4,5] &= ([1,2,3] +\!\!+ [4]) +\!\!+ [5]
\end{aligned}
$$

Zašto su korisne funkcije $foldr, foldl$?

- Neke rekurzivne funkcije nad listama se definišu na jednostavniji način.
- Svojstva funkcija definisanih pomoću $foldr/foldl$ se mogu dokazati algebarskim tehnikama.
- Napredniji programi mogu biti efikasniji ako se umjesto eksplicite rekurzije koristi $foldr/foldl$.

Operator kompozicije

Šta se dobija kao rezultat?

- Kompozicija dvije funkcije:

$$\begin{array}{lcl} (\circ) & :: & (b \to c) \to (a \to b) \to (a \to c) \\ f \circ g & = & \lambda x \to f\ (g\ x) \end{array}$$

$$(f \circ g)\ x = f\ (g\ x)$$

- Primjeri:

$$\begin{array}{lcl} odd\ n & = & \neg\ (even\ n) \\ twice\ f\ x & = & f\ (f\ x) \\ sumsqreven\ ns & = & sum\ (map\ (\uparrow 2)\ (filter\ even\ ns)) \end{array}$$

$$\begin{array}{lcl} odd & = & \neg \circ even \\ twice\ f & = & f \circ f \\ sumsqreven & = & sum \circ map\ (\uparrow 2) \circ filter\ even \end{array}$$

Operator kompozicije

Tipovi funkcija?

- Kompozicija je asocijativna.

$$f \circ (g \circ h) = (f \circ g) \circ h$$

$$id \circ f = f \text{ and } f \circ id = f$$

- U ghci:

```
odd :: Int → Bool
odd  = not . even
```

Zadaci:

72. Upotrebom neke od funkcija $fold$, definisati funkciju koja za argument uzima listu stringova i spaja ih u jedan string, u kojem su „početni stringovi odvojeni znakom za novu liniju ‘\n’. Na primjer, za listu [“danas”, ”juce”, ”sutra”], funkcija treba da kao rezultat vrati string „danas\n juce\n sutra“.

73. Definisati funkciju koja za argument uzima string, koji se sastoji od više linija (kraj svake linije je označen znakom ‘\n’), a kao rezultat vraća prvu liniju. Na primjer, za string „danas je ponedjeljak\n juce je bila nedjelja\n sutra je utorak“, funkcija vraća string „danas je ponedjeljak”. Funkciju definisati:
    - rekurzivno;
    - pomoću funkcija višeg reda, a potrebne pomoćne funkcije definisati kao lambda funkcije.

74. Definisati funkciju koja za argument uzima string, koji se sastoji od više linija (kraj svake linije je označen znakom ‘\n’), a kao rezultat vraća string iz kojeg je izbačena prva linija. Na primjer, za string „danas je ponedjeljak\n juce je bila nedjelja\n sutra je utorak“, funkcija vraća string „juce je bila nedjelja\n sutra je utorak”. Funkciju definisati:
    - rekurzivno;
    - pomoću funkcija višeg reda, a potrebne pomoćne funkcije definisati preko lambda funkcija.

Za vježbu:

- 7.6 – String transmitter problem;
- Zadaci 7.7;

DEKLARISANJE TIPOVA I KLASA

Haskell

Deklaracija **type**

- Deklarisanje tipa uvođenjem novog imena za postojeći tip – **type**.

$$\textbf{type}\ \mathit{String} = [\mathit{Char}]$$

- Ime tipa počinje velikim slovom.
- Deklaracija tipa može biti ugnježdena.

$$\begin{aligned}\textbf{type}\ \mathit{Board} &= [\mathit{Pos}]\\ \textbf{type}\ \mathit{Pos} &= (\mathit{Int}, \mathit{Int})\end{aligned}$$

```
type Pos = (Int,Int)

type Trans = Pos -> Pos
```

Deklaracija **type**

- Deklaracija tipa ne može biti rekurzivna.

$$\textbf{type}\ Tree \quad = \quad (Int, [Tree])$$

- Deklaracija tipa može biti parametrizovana drugim tipom.

```
type Pair a = (a,a)
```

$$\textbf{type}\ Assoc\ k\ v \quad = \quad [(k, v)]$$

$$\begin{aligned} find &:: Eq\ k \Rightarrow k \rightarrow Assoc\ k\ v \rightarrow v \\ find\ k\ t &= head\ [v \mid (k', v) \leftarrow t, k == k'] \end{aligned}$$

Deklaracija **data**

- Deklarisanje potpuno novog tipa – **data**.

$$\textbf{data}\ Bool \quad = \quad False \mid True$$

- „or“; konstruktori tipa;
  - imena konstruktora moraju počinjati velikim slovom;
- Upotreba vrijednosti novog tipa?

Zadaci:

75. Definisati podatak $Move$ koji ima vrijednosti $Left, Right, Up, Down$, kao i podatak Pos koji predstavlja tačku iz dvodimenzionalnog cjelobrojnog prostora. Napisati funkcije:
    - koja pomjera tačku u odnosu na vrijednost $Move$;
    - koja za listu uzima listu vrijednosti tipa $Move$ i jednu vrijednost Pos, a kao rezultat vraća Pos u kojoj ce biti tačka nakon što se pomjeri po svim elemetima date liste;
    - koja vraća suprotnu vrijednost vrijednosti $Move$;

Deklaracija **data** – rješenje zadatka 1.

$$\textbf{data}\ Move \quad = \quad Left \mid Right \mid Up \mid Down$$

$$
\begin{array}{lcl}
move & :: & Move \to Pos \to Pos \\
move\ Left\ (x, y) & = & (x - 1, y) \\
move\ Right\ (x, y) & = & (x + 1, y) \\
move\ Up\ (x, y) & = & (x, y - 1) \\
move\ Down\ (x, y) & = & (x, y + 1) \\
moves & :: & [Move] \to Pos \to Pos \\
moves\ [\,]\ p & = & p \\
moves\ (m : ms)\ p & = & moves\ ms\ (move\ m\ p) \\
flip & :: & Move \to Move \\
flip\ Left & = & Right \\
flip\ Right & = & Left \\
flip\ Up & = & Down \\
flip\ Down & = & Up
\end{array}
$$

Deklaracija **data**

- Konstruktori tipa takođe mogu imati parametre.

$$\textbf{data}\ Shape \quad = \quad Circle\ Float \mid Rect\ Float\ Float$$

Zadaci:

76. Napisati funkcije:
    - koja vraća kvadrat stranice n;
    - Koja računa površinu oblika;

Deklaracija **data**

- Konstruktori kao funkcije.

$$
\begin{aligned}
&> :\mathbf{type}\ Circle \\
&Circle :: Float \rightarrow Shape \\
\\
&> :\mathbf{type}\ Rect \\
&Rect :: Float \rightarrow Float \rightarrow Shape
\end{aligned}
$$

Deklaracija **data**

- Deklaracija tipa može biti parametrizovana.

$$\textbf{data}\ Maybe\ a \quad = \quad Nothing \mid Just\ a$$

$$\begin{aligned} safediv \quad & :: \quad Int \rightarrow Int \rightarrow Maybe\ Int \\ safediv\ \_\ 0 \quad & = \quad Nothing \\ safediv\ m\ n \quad & = \quad Just\ (m\ `div`\ n) \end{aligned}$$

Zadaci:

77. Koristeći $Maybe$ tip definisati funkciju $safehead$, koja vraća glavu neprazne liste, a u suprotnom ništa.

Zadaci:

78. Definisati novi tip podatka koji predstavlja radne dane u sedmici i napisati funkciju koja za argument uzima podatak tipa dan u sedmici i vraća informaciju da li je u pitanju radni dan.

79. Definisati korisnički tip podataka koji predstavlja listu telefonskih poziva. ListaPoziva se sastoji od uređenih parova, gdje prvi element para predstavlja poziv na određen broj ili prekid poziva, a drugi element predstavlja vrijeme kada se desio događaj opisan prvim elementom. Napisati funkciju listing koja datu ListuPoziva transformiše u listu koja sadrži uređene trojke, gdje svaki zapis ima tri podatka: pozivani broj, početak i kraj poziva. Može se pretpostaviti da je data ListaPoziva ispravna, odnosno da ako je jedan element liste poziv na broj, sljedeći je prekid. Pretpostavka važi za sve, osim za posljednji element u listi.

Rekurzivni tipovi

- Novi tipovi se mogu definisati i rekurzivno.
- Prirodni brojevi

$$\textbf{data}\ Nat = Zero \mid Succ\ Nat$$

$$Zero$$
$$Succ\ Zero$$
$$Succ\ (Succ\ Zero)$$
$$Succ\ (Succ\ (Succ\ Zero))$$
$$\vdots$$

```
Succ (Succ (Succ Zero))
```

```
1 + (1 + (1 + 0))  =  3
```

Zadaci:

80. Definisati sljedeće konverzione funkcije:
    - Funkciju koja pretvara Nat u Int;
    - Funkciju koja pretvara Int u Nat;

Rekurzivni tipovi

- Konverzione funkcije - rješenja;

```
nat2int           :: Nat → Int
nat2int Zero       = 0
nat2int (Succ n) = 1 + nat2int n


int2nat  :: Int → Nat
int2nat 0 = Zero
int2nat n = Succ (int2nat (n-1))
```

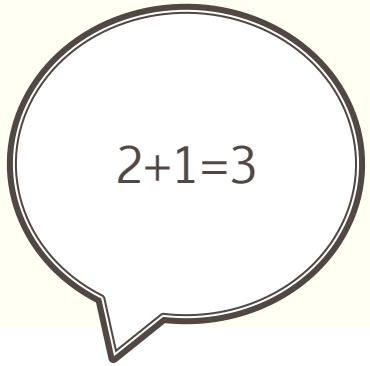
Zadaci:

81. Definisati funkciju koja sabira 2 broja, koji su podaci tipa Nat.

Rekurzivni tipovi

- Sabiranje dva broja.

$$
\begin{array}{lcl}
add & :: & Nat \rightarrow Nat \rightarrow Nat \\
add\ m\ n & = & int2nat\ (nat2int\ m + nat2int\ n)
\end{array}
$$



$$
\begin{array}{lcl}
add\ Zero\ n & = & n \\
add\ (Succ\ m)\ n & = & Succ\ (add\ m\ n)
\end{array}
$$

$$
\begin{array}{ll}
 & add\ (Succ\ (Succ\ Zero))\ (Succ\ Zero) \\
= & \quad \{\ \text{applying } add\ \} \\
 & Succ\ (add\ (Succ\ Zero)\ (Succ\ Zero)) \\
= & \quad \{\ \text{applying } add\ \} \\
 & Succ\ (Succ\ (add\ Zero\ (Succ\ Zero))) \\
= & \quad \{\ \text{applying } add\ \} \\
 & Succ\ (Succ\ (Succ\ Zero))
\end{array}
$$

Rekurzivni tipovi

- Lista proizvoljnih elemenata

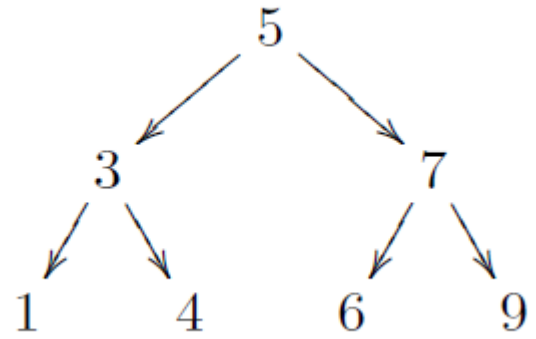
$$\textbf{data}\ List\ a \quad = \quad Nil \mid Cons\ a\ (List\ a)$$

$$\begin{array}{lcl} len & :: & List\ a \rightarrow Int \\ len\ Nil & = & 0 \\ len\ (Cons\ \_\ xs) & = & 1 + len\ xs \end{array}$$

Rekurzivni tipovi

- Binarno stablo
  - Vrste čvorova?

Stablo proizvoljnog tipa?



$$\textbf{data}\ \mathit{Tree} \quad = \quad \mathit{Leaf}\ \mathit{Int} \mid \mathit{Node}\ \mathit{Tree}\ \mathit{Int}\ \mathit{Tree}$$

$$\begin{aligned} t &:: \mathit{Tree} \\ t &= \mathit{Node}\ (\mathit{Node}\ (\mathit{Leaf}\ 1)\ 3\ (\mathit{Leaf}\ 4))\ 5\ (\mathit{Node}\ (\mathit{Leaf}\ 6)\ 7\ (\mathit{Leaf}\ 9)) \end{aligned}$$

Zadaci:

82. Napisati funkciju koja za argument uzima broj i binarno stablo brojeva i vraća informaciju da li se dati broj pojavljuje u datom stablu.
83. Napisati funkciju koja za argument uzima binarno stablo brojeva i vraća listu brojeva koji se pojavljuju u tom binarnom stablu.
84. Napisati funkciju koja za argument uzima broj i uređeno binarno stablo (binarno stablo pretrage) brojeva i vraća informaciju da li se dati broj pojavljuje u datom stablu.

Rekurzivni tipovi

- Različite rekurzivne definicije stabala:

Razlike?

$$
\begin{aligned}
&\textbf{data}\ Tree\ a &&= Leaf\ a \mid Node\ (Tree\ a)\ (Tree\ a)\\
&\textbf{data}\ Tree\ a &&= Leaf \mid Node\ (Tree\ a)\ a\ (Tree\ a)\\
&\textbf{data}\ Tree\ a\ b &&= Leaf\ a \mid Node\ (Tree\ a\ b)\ b\ (Tree\ a\ b)\\
&\textbf{data}\ Tree\ a &&= Node\ a\ [Tree\ a]
\end{aligned}
$$

Koji čvor je list u posljednjoj definiciji?

Zadaci:

85. Definisati korisnički tip podataka koji predstavlja kartu (za igranje karata) i ruku koja sadrži više karata. Zatim, napisati funkciju koja za argument uzima ruku i kartu, pa iz date ruke bira kartu koja pobjeđuje zadatu kartu, po sljedećem principu:
    - pratimo boju karte;
    - ako u ruci postoji karta koja pobjeđuje zadanu onda se bira ta karta;
    - u suprotnom, bira se najslabija karta.

Deklaracija klasa i instanci

- Definisanje novih klasa – **class**.

```
class Eq a where
    (==), (≠)   ::   a → a → Bool
    x ≠ y       =    ¬ (x == y)
```

```
instance Eq Bool where
    False == False   =   True
    True == True     =   True
    _ == _           =   False
```

Samo tipovi deklarisani sa **data**!

override default definicija;

Deklaracija klasa i instanci

- Klasa kao proširenje postojeće klase.

$$
\begin{array}{lll}
\textbf{class } Eq\ a \Rightarrow Ord\ a \textbf{ where} & & \\
\quad (<), (\leqslant), (>), (\geqslant) & :: & a \to a \to Bool \\
\quad min, max & :: & a \to a \to a \\
\quad min\ x\ y \mid x \leqslant y & = & x \\
\qquad\qquad \mid otherwise & = & y \\
\quad max\ x\ y \mid x \leqslant y & = & y \\
\qquad\qquad \mid otherwise & = & x
\end{array}
$$

Deklaracija klasa i instanci

- Klasa kao proširenje postojeće klase.

instance *Ord Bool* **where**

$$
\begin{aligned}
False < True &= True \\
\_ < \_ &= False \\
b \leqslant c &= (b < c) \vee (b == c) \\
b > c &= c < b \\
b \geqslant c &= c \leqslant b
\end{aligned}
$$

Derived instance

- Mehanizam za proglašavanje da su novi tipovi klase $Eq, Ord, Show, Read$ – **deriving**.

```
data Bool = False | True
          deriving (Eq, Ord, Show, Read)
```

```
> False == False
True

> False < True
True

> show False
"False"

> read "False" :: Bool
False
```

Derived instance

- Konstruktori sa argumentima – argumenti pripadaju klasi;

$$\begin{aligned}&\textbf{data}\ Shape &&= Circle\ Float \mid Rect\ Float\ Float\\ &\textbf{data}\ Maybe\ a &&= Nothing \mid Just\ a\end{aligned}$$

$$\begin{aligned}&> Rect\ 1.0\ 4.0 < Rect\ 2.0\ 3.0\\ &True\\ \\ &> Rent\ 1.0\ 4.0 < Rect\ 1.0\ 3.0\\ &False\end{aligned}$$

Zadaci:

86. Dati su sljedeći tipovi podataka:

$$data\ Clan\ =\ Clan\ String\ String$$
$$data\ Knjiga\ =\ Naslov\ String$$
$$type\ Kartica\ =\ (Clan, Knjiga)$$
$$type\ Baza\ =\ [Kartica]$$

koji redom predstavljaju člana biblioteke, knjigu u biblioteci, karticu koja sadrži informaciju da dati član duži datu knjigu i listu svih kartica u biblioteci. Napisati sljedeće funkcije:

a) $dužnici$ – koja za argumente uzima bazu i knjigu, a kao rezultat vraća listu članova baze koji duže datu knjigu;

b) $knjige$ – koja za argument uzima bazu i člana, a kao rezultat vraća listu knjiga koje je zadužio dati član u datoj bazi;

Zadaci:

c) $zadužena$ – koja za argument uzima bazu i knjigu, a kao rezultata vraća $True$ ako je data knjiga zadužena u datoj bazi, inače $False$;

d) $brojPrimjeraka$ - koja za argument uzima bazu i knjigu, a kao rezultata vraća informaciju koliko primjeraka date knjige je iznajmljeno u datoj bazi;

e) $izdaj$ – koja za argument uzima člana, knjigu i bazu i bilježi informaciju da je datom članu izdata data knjiga u datoj bazi;

f) $vrati$ - koja za argument uzima člana, knjigu i bazu i bilježi informaciju da je dati član vratio datu knjigu u datu bazu.

Zadaci:

87. Definisati korisnički podatka koji predstavlja Tačku u prostoru $\mathbb{R}^2$ ($data\ Tacka = Tacka\ Float\ Float$). Put se definiše kao lista Tačaka. Napisati sljedeće funkcije:
    - $koordinataX$ – koja vraća prvu koordinatu Tačke;
    - $koordinataY$ – koja vraća drugu koordinatu Tačke;
    - $udaljenost$ – koja računa rastojanje između dvije Tačke;
    - $dužinaPuta$ – koja računa dužinu datog Puta kao sumu rastojanja između njegovih uzastopnih Tačaka.

Za vježbu:

- 10.4 Tautology checker (strana 125);
- 10.8 Zadaci za vježbu (strana 135);
- 99 problems in Haskell:
  - https://wiki.haskell.org/H-99:_Ninety-Nine_Haskell_Problems

INTERAKTIVNI PROGRAMI

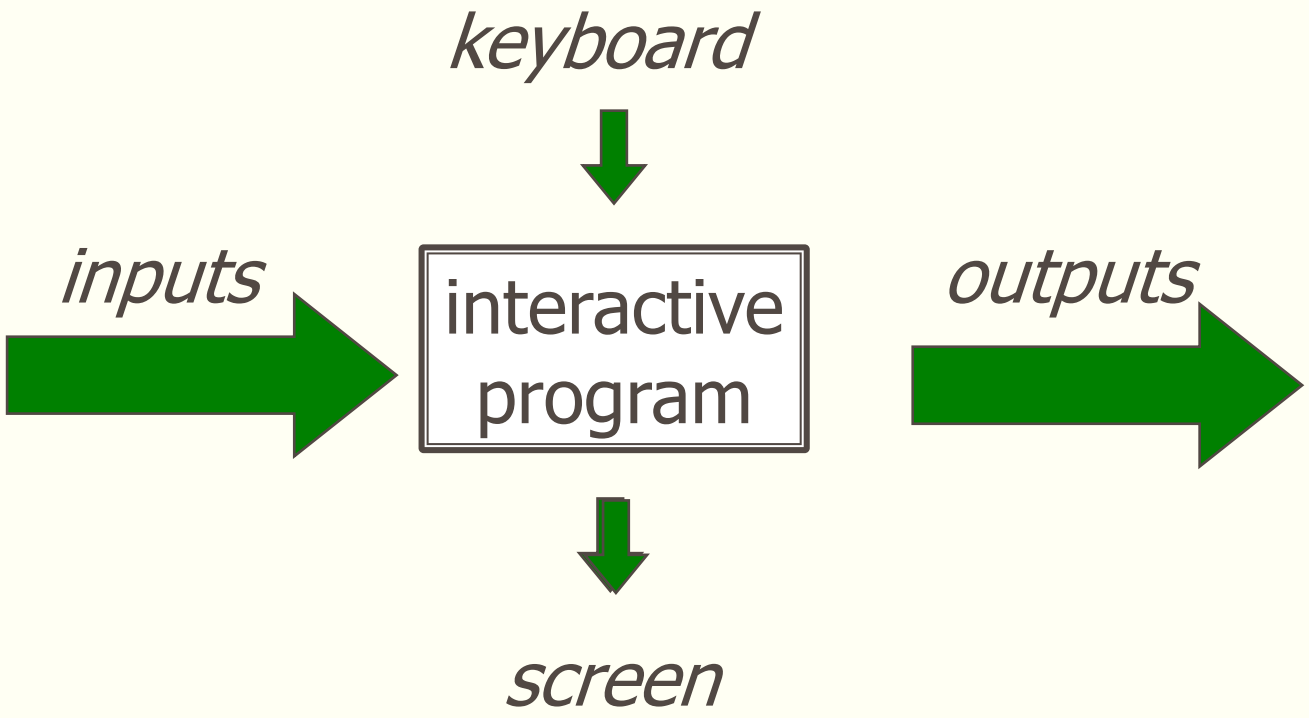
Interakcija

- batch programi;

- Do sada napisati kodovi u Haskellu;
- Kompajleri: `Prog -> Code`

Interakcija

- interaktivni programi;



Interakcija

- Pisanje interaktivnih programa u Haskellu?
  - Problemi u Haskell-u se rješavaju pisanjem čisto matematičkih funkcija.
  - Bočni efekat?
  - Razna rješenja tokom godina;
  - Novi tip u konjukciji sa malim brojem primitiva;

Ulazno/Izlazni tip

- Funkcija;

$$\textbf{type } IO \quad = \quad World \rightarrow World$$

$$\textbf{type } IO\ a \quad = \quad World \rightarrow (a, World)$$

- IO – akcije: IO a, gdje je a tip povratne vrijednosti akcije;

Ulazno/Izlazni tip

```
IO Char
```

Tip akcije koja vraća karakter.

```
IO ()
```

Tip akcije koja nema povratnu vrijednost.

Ulazno/Izlazni tip

- Interaktivni programi mogu da uzimaju argumente;
  - Primjer:

$$Char \rightarrow IO\ Int,$$

$$Char \rightarrow World \rightarrow (Int, World).$$

- IO implementacija:

```
data IO a = ...
```

Osnovne akcije

- *getChar*

```
getChar :: IO Char
```

- *putChar*

```
putChar :: Char → IO ()
```

- *return*

```
return :: a → IO a
```

Sekvencijalni niz IO akcija

- Više akcija u okviru jedne akcije;

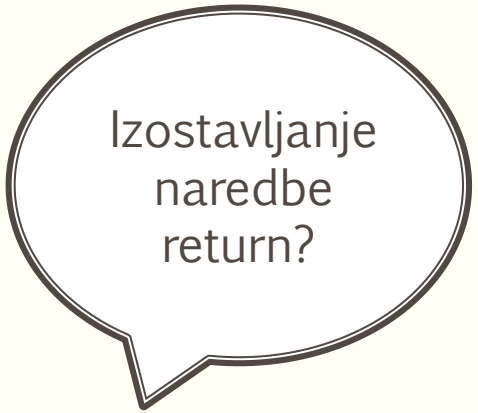
```
do v1 <- a1
   v2 <- a2
   .
   .
   .
   vn <- an
   return (f v1 v2 ... vn)
```

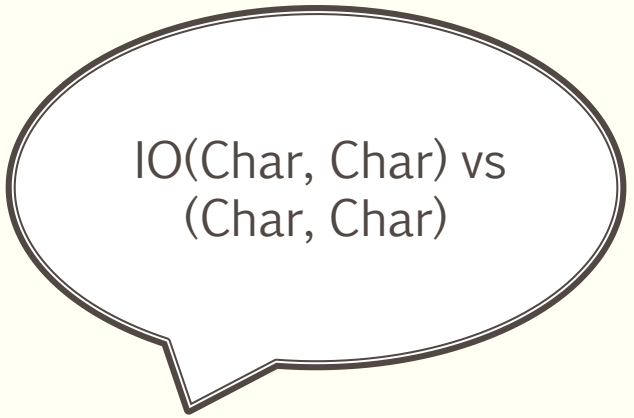
- Pravila:
  - Poravnanje;
  - Generatori;
  - a_i u značenju _ $\leftarrow a_i$;

Sekvencijalni niz IO akcija

- Šta radi naredna funkcija?

```
act :: IO (Char,Char)
act = do x ← getChar
         getChar
         y ← getChar
         return (x,y)
```





Izvođenje primitiva

- Čitanje Stringa sa tastature;

```
getLine :: IO String
getLine = do x ← getChar
             if x == '\n' then
                return []
             else
                do xs ← getLine
                   return (x:xs)
```

Izvođenje primitiva

- Ispisivanje Stringa na ekran;

```
putStr :: String → IO ()
putStr []     = return ()
putStr (x:xs) = do putChar x
                   putStr xs
```

```
putStrLn :: String → IO ()
putStrLn xs = do putStr xs
                 putChar '\n'
```

Zadaci:

88. Napisati funkciju koja čita dva Stringa sa ulaza i ispisuje ih na ekranu.
89. Napisati funkciju koja čita String sa tastature i na ekranu ispisuje njegovu dužinu.
90. Napisati funkciju koja čita sadržaj txt fajla i ispisuje ga na ekran.
91. Napisati funkciju koja izvršava više interaktivnih naredbi, koje se zadaju listom.
92. Napisati program koji definiše aritmetičke izraze, koji se sastoje od cijelih brojeva i operacija sabiranja i množenja. Program treba da sadrži funkciju koja računa vrijednost aritmetičkog izraza, funkciju za ispis aritmetičkog izraza na ekran. Kao i funkciju koja prikazuje korisniku izraz, omogućava da unese vrijednost izraza i vraća informaciju da li je unesena vrijednost jednaka vrijednosti izraza.

Zadaci za vježbu:

- 9.6 Kalkulator
- 9.7 Igra život;
- 10.7 Igra NIM;

THE COUNTDOWN PROBLEM

Countdown problem

- https://www.youtube.com/watch?v=CiXDS3bBBUo&list=PLF1Z-APd9zK7usPMx3LGMZEHrECUGodd3&index=13

Napredni nivo!

- Drugo izdanje knjige;
- Funktori, Monade
  - https://www.youtube.com/playlist?list=PLF1Z-APd9zK5uFc8FKr_di9bfsYv8-lbc
  - https://www.youtube.com/watch?v=t1e8gqXLbsU

PROJEKAT

Ideje

- ...

Literatura:

- Introduction to functional programing (John Harrison)
- Programing in Haskell (Graham Huton)
- Funkcionalno programiranje – prednosti i nedostaci (Nenad Mitić)