Chapter 1

Funcții de ordin superior

În Haskell (și în orice limbaj funcțional) *funcțiile* joacă un rol central și considerate obiecte (nu în sensul POO) de prim rang (first-class).

În acest sens, în Haskell funcțiile pot fi, de exemplu:

1. create în momentul rulării;
2. transmise ca argumente unor alte funcții;
3. returnate ca rezultat al unei funcții.

Cu alte cuvinte, în limbajele funcționale, valorile obișnuite (de tip întreg sau bool) nu au alt statut și nu sunt tratate diferit față de funcții.

Într-un limbaj de nivel scăzut, cum ar fi C sau Java, utilizarea funcțiilor este limitată. De exemplu, deși este foarte ușor să trimit un `int` ca argument, nu pot trimite o funcție ca argument.

Funcțiile care nu primesc funcții ca argument și nici nu întorc funcții se numesc funcții de ordinul I.

Funcțiile care primesc o funcție ca argument sau întorc o funcție ca argument se numesc funcții de ordin superior (higher-order functions).

Limbajele de nivel scăzut, cum ar fi C++ sau Java, permit definirea doar de funcții de ordinul I [1].

1.1 Funcții care primesc funcții ca argument

```
add2 :: Integer -> Integer
add2 x = x + 2

mult3 :: Integer -> Integer
mult3 x = x * 2

twice f x = f (f x)
```

```
ghci> twice add2 3
ghci> twice mult3 7
ghci> :t twice
```

Atenție! Funcțiile nu pot fi transformate în String:

```
ghci> show add2
```

[1] Pot fi simulate funcțiile de ordin superior folosind obiecte.

1.2 Funcții care întorc funcții ca argument

```
adder c = \x -> x + c

add2 = adder 2
add3 = adder 3

ghci> add2 42
ghci> add3 42
ghci> (adder 7) 42
```

1.3 Funcții de mai multe argumente în Haskell

```
ghci> :t adder
```

Syntactic sugar: `a -> b -> c` este o prescurtare pentru `a -> (b -> c)`, iar `f x y` este o prescurtare pentru `(f x) y`.

1.4 Funcții cu 0 argumente

```
aaa :: Integer
aaa = 7

bbb :: Integer
bbb = bbb

ghci> aaa
ghci> bbb
```

1.5 Exemple de funcții de ordin superior

```
reduce :: (b -> a -> b) -> b -> [a] -> b
reduce _ init [] = init
reduce f init (x:xs) = reduce f (f init x) xs

ghci> reduce [1,2,3,4] 0 (+)
ghci> :t (.)
ghci> :t ($)
ghci> :t (map)
ghci> :t (filter)
ghci> :t (foldl)
```