

Laboratorul 1: Introducere în Haskell

Pentru început, vă veți familiariza cu mediul de programare GHC (Glasgow Haskell Compiler). Acesta include doua componente: **GHCi** (care este un interpretor) și **GHC** (care este un compilator).

Descărcare și instalare

Pentru instalare puteți citi mini-tutorialul de la acest [https://docs.google.com/document/d/1lMvx4dRw1rXQ1KiW80poZJwG6F0v6FQU/edit\[LINK\]](https://docs.google.com/document/d/1lMvx4dRw1rXQ1KiW80poZJwG6F0v6FQU/edit[LINK])

De asemenea, este recomandată folosirea unui stil standard de formatare a fișierelor sursă, spre exemplu <https://github.com/tibbe/haskell-style-guide/blob/master/haskell-style.md>.

GHCi

1. Deschideți un terminal si introduceți comanda **ghci** (în Windows este posibil să aveți instalat WinGHCi). După câteva informații despre versiunea instalată va apare

Prelude>

Prelude este librăria standard: <http://hackage.haskell.org/package/base-4.12.0.0/docs/Prelude.html>

În interpretor puteți:

- să introduceți expresii, care vor fi evaluate atunci cand este posibil:

```
Prelude> 2+3
5
Prelude> False || True
True
Prelude> x
<interactive>:10:1: error: Variable not in scope: x
Prelude> x=3
Prelude> x
3
Prelude> y=x+1
Prelude> y
4
Prelude> head [1,2,3]
1
Prelude> head "abcd"
'a'
Prelude> tail "abcd"
'bcd'
```

Funcțiile `head` și `tail` aparțin modulului standard **Prelude**.

- să introduceți comenzi, orice comandă fiind precedată de ":"

:? - este comanda *help*

:q - este comanda *quit*

:cd - este comanda *change directory*

:t - este comanda *type*

```
Prelude> :t True
```

```
True :: Bool
```

Cititi mai mult despre **GHCi**:

https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/ghci.html

Fișiere sursă

- Fișierele sursă sunt fișiere cu extensia **.hs**, pe care le puteți edita cu un editor la alegerea voastră. Deschideți fișierul **lab1.hs** care conține următorul cod:

[illegible]

Fără încărcarea fișierului, încercați să calculați `double myInt`:

```
Prelude> double myInt
```

Observați mesajele de eroare. Acum încărcați fișierul folosind comanda **load** (:!) și încercați din nou să calculați `double myInt`:

```
Prelude> :l lab1.hs
[1 of 1] Compiling Main                ( lab1.hs, interpreted )
Ok, 1 module loaded.
*Main> double myInt
*Main> double 2000
```

Modificați fișierul adăugînd o funcție `triple`. Dacă fișierul este deja încărcat, puteți să îl reîncărați folosind comanda ***reload*** (`:r`).

Puteti reveni în **Prelude** folosind :m -

```
Prelude> :l lab1.hs
[1 of 1] Compiling Main                ( lab1.hs, interpreted )
Ok, 1 module loaded.
*Main> :m - Main
Prelude>
```

Ați observat că în mesajele primite a apărut noțiunea de *modul*. Practic, fișierul lab1.hs conține un modul care se numește **Main**, definit automat.

Elemente de limbaj

3. Există numeroase librării foarte utile. Cum putem să le identificăm? O sursă de informații foarte bună este **Hoogle** <https://hoogle.haskell.org/>

Căutați funcția `head` folosită anterior. Observăm că se găsește atât în librăria `{Prelude}`, cât și în librăria `Data.List`.

Să presupunem că vrem să generăm toate permutările unei liste. Căutați funcția `permutation` (sau ceva asemănător) și observăm că în librăria `Data.List` se găsește o funcție `permutations`. Faceți click pe numele funcției (sau al librăriei) pentru a putea citi detalii despre această funcție. Pentru a o folosi în interpretor va trebui să încărcăm librăria `Data.List` folosind comanda `import`

```
Prelude> :t permutations
<interactive>:1:1: error: Variable not in scope: permutations
Prelude> import Data.List
Prelude Data.List> :t permutations
permutations :: [a] -> [[a]]
Prelude Data.List> permutations [1,2,3]
[[1,2,3],[2,1,3],[3,2,1],[2,3,1],[3,1,2],[1,3,2]]
Prelude Data.List> permutations "abc"
["abc","bac","cba","bca","cab","acb"]
```

Atenție! funcția `permutations` întoarce o listă de liste.

Eliminați librăria folosind

```
Prelude> :m - Data.List
```

Librăriile se includ în fișiere sursă folosind comanda `import`. Descideți fișierul `lab1.hs` și adugați la început

```
import Data.List
```

Încărcați fișierul în interpretor și evaluați

```
*Main> permutations [1..myInt]
```

Ce se întâmplă? `[1..myInt]` este lista `[1,2,3,..., myInt]` care are o dimensiune foarte mare. Observăm că putem folosi valori numerice foarte mari. Evaluarea expresiei o oprim cu `Ctrl+C`.

În librăria `Data.List` căutați funcția `subsequences`, înțelegeți ce face și folosiți-o pe câteva exemple.

Indentare

4. În Haskell se recomandă scrierea codului folosind *indentarea*. În anumite situații, nerespectarea regulilor de indentare poate provoca erori la încărcarea programului.

În fișierul `lab1.hs` deplasați cu câteva spații definiția funcției `double`:

```
double :: Integer -> Integer
double x = x+x
```

Reîncărcați programul. Ce observați?

Atenție! În unele editoare se recomandă înlocuirea tab-urilor cu spații.

Să definim funcția `maxim`

```
maxim :: Integer -> Integer -> Integer
maxim x y = if (x > y) then x else y
```

Varianta cu indentare este:

```
maxim :: Integer -> Integer -> Integer
maxim x y =
    if (x > y)
        then x
        else y
```

Dorim acum să scriem o funcție care calculează maximul a trei numere. Evident, o varianta este

```
maxim3 x y z = maxim x (maxim y z)
```

Scrieți funcția `maxim3` fără a folosi `maxim`, utilizând direct `if` și scrierea indentată.

Putem scrie funcția `maxim3` folosind expresia `let...in` astfel

```
maxim3 x y z = let u = (maxim x y) in (maxim u z)
```

Atenție! expresia `let...in` crează scop local.

Varianta cu indentare este

```
maxim3 x y z =
    let
        u = maxim x y
    in
        maxim u z
```

Scrieți o funcție `maxim4` folosind varianta cu `let...in` și indentare.

Scrieți o funcție care testează funcția `maxim4` prin care să verificați ca rezultatul este în relația `>=` cu fiecare din cele patru argumente (operatorii logici în Haskell sunt `||`, `&&`, `not`).

Citiți mai multe despre indentare <https://en.wikibooks.org/wiki/Haskell/Indentation>

Tipuri de date

5. Din exemplele de până acum ați putut observa că în Haskell:

a) există tipuri predefinite: `Integer`, `Bool`, `Char`

b) se pot construi tipuri noi folosind `[]`

```
*Main> :t [1..myInt]
[1..myInt] :: [Integer]

Prelude> :t "abc"
"abc" :: [Char]
```

Evident, `[a]` este tipul *listă de date de tip a*. Tipul `String` este un sinonim pentru `[Char]`.

c) Ați întâlnit tipul `Bool` și valorile `True` și `False`. În Haskell tipul `Bool` este definit astfel

```
data Bool = False | True
```

În această definiție, `Bool` este un *constructor de tip*, iar `True` și `False` sunt *constructori de date*.

d) Sistemul tipurilor în Haskell este mult mai complex. Fără a încărca fișierul `lab1.hs`, definiți direct în GHCi funcția `maxim`:

```
Prelude> maxim x y = if (x > y) then x else y
```

Cu ajutorul comenzii `:t` aflați tipul acestei funcții. Ce observați?

```
Prelude> :t maxim  
maxim :: Ord p => p -> p -> p
```

Răspunsul primit trebuie interpretat astfel: `p` reprezintă un tip arbitrar înzestrat cu o relație de ordine, funcția `maxim` are două argumente de tip `p` și întoarce un rezultat de tip `p`.

Astfel, tipul unei operații poate fi definit de noi sau dedus automat. Vom discuta mai multe în cursurile și laboratoarele următoare.

Exerciții

6. Să se scrie următoarele funcții:

- funcție cu 2 parametri care calculează suma pătratelor celor două numere;
- funcție cu un parametru ce întoarce mesajul “par” dacă parametrul este par și “impar” altfel;
- funcție care calculează factorialul unui număr;
- funcție care verifică dacă un primul parametru este mai mare decât dublul celui de-al doilea parametru.

Material suplimentar

- Citiți capitolul *Starting Out* din M. Lipovaca, Learn You a Haskell for Great Good! <http://learnyouahaskell.com/starting-out>