

## Laborator 1

### Introducere în Haskell

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

(L1.1) [GHCi] Deschideți un terminal și 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 când 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
```

*Pe versiunile mai vechi de Haskell, în loc de `x = 3` trebuie folosit `let x = 3`.*

```
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

Citiți mai mult despre GHCI:

[https://downloads.haskell.org/~ghc/latest/docs/html/users\\_guide/ghci.html](https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/ghci.html)

**(L1.2) [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]

```
double  :: Integer -> Integer
```

```
double x = x+x
```

Fără încărcarea fişierul, încercați să calculati double myInt:

```
Prelude> double myInt
```

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

```
Prelude> :l lab1.hs
```

[1 of 1] Compiling Main

$$( \text{lab1.hs}, \text{interpreted} )$$

Ok, 1 module loaded.

```
*Main> double myInt
```

```
*Main> double 2000
```

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

Puteți 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. Despre module vom discuta mai târziu.

**În continuare vom discuta câteva elemente de limbaj.**

**(L1.3) [Hoogle. Librării]** 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 asemanător) 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.

**(L1.4) [Indentare]** 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 bifarea opțiunii de înlocuire a `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>

**(L1.5) [Tipuri]** Din exemplele de până acum ați putut observa că în Haskell:

- există tipuri predefinite: `Integer`, `Bool`, `Char`
- 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]`.

- 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*. În exercițiul următor vom defini un tip de date nou într-un mod similar.

- 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.

**(L1.6) [Piatra, Foarfeca, Hartie]** În acest exercițiu vom modela jocul Piatra, Foarfeca, Hartie astfel:

- fișierul `pfh.hs` conține un tip de date `Alegere`

```
data Alegere  
  = Piatra  
  | Foarfeca  
  | Hartie  
deriving (Eq, Show)
```

Precizarea `deriving (Eq, Show)` ne spune ca pentru datele de acest tip este definită în mod natural relația de egalitate și că pot fi afișate ca șir de caractere.

- definiți în mod similar tipul de date `Rezultat` cu valorile `Victorie`, `Infrangere` și `Egalitate`.

Pentru toate datele de mai sus, verificați tipul folosind `:t`.

- Scrieți funcția

`partida :: Alegere -> Alegere -> Rezultat`

care întoarce `Victorie` când primul argument este victorios, `Infrangere` când primul argument pierde și `Egalitate` când argumentele sunt egale.

## Material suplimentar

Pentru a folosi Haskell, este recomandată folosirea uneltei `Haskell Stack`. Puteți citi mai multe despre cum se instalează și folosește la <https://haskell-lang.org/get-started>.

Este recomandată folosirea `ghcid` ([http://www.parsonsmatt.org/2018/05/19/ghcid\\_for\\_the\\_win.html](http://www.parsonsmatt.org/2018/05/19/ghcid_for_the_win.html)) în paralel cu un editor de text.

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>.

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