Laboratorul 1: Introducere în Haskell

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

Descărcare și instalare

Pentru instalare, citiți mini-tutorialul de la adresa:
https://docs.google.com/document/d/1lMvx4dRw1rXQ1KiW80poZJwG6F0v6FQU/edit

Recomandăm folosirea unui stil standard de formatare a fișierelor sursă, precum cel de la adresa:
https://github.com/tibbe/haskell-style-guide/blob/master/haskell-style.md

GHCi

1. Deschideți un terminal și introduceți comanda `ghci` (în Windows este posibil să aveți instalat WinGHCi). După câteva informații despre versiunea instalată, va apărea promptul:

```
ghci>
```

sau, în funcție de versiunea instalată:

```
Prelude>
```

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

În interpretor puteți:

- introduce 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
Prelude> head [1,2,3]
1
Prelude> head "abcd"
'a'
```

```
Prelude> tail "abcd"
'bcd'
```

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

- introduce comenzi; orice comandă este 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

Fișiere sursă

2. Fișierele sursă sunt fișiere text cu extensia ***.hs***. Le puteți edita cu un editor la alegerea voastră. Deschideți fișierul `lab1.hs` care conține următoarele linii de cod:

```
myInt = 3141592653589793238462643383279502884197169399375105820974944592 3
double :: Integer -> Integer
double x = x+x
```

Fără a încărca fișierul, încercați să calculați `double myInt`:

```
Prelude> double myInt
```

Observați mesajele de eroare. Acum încărcați fișierul folosind comanda ***load*** (:l).

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

Promptul poate rămâne neschimbat, sau să fie înlocuit cu numele unui ***modul***. De exemplu, în linia următoare, este înlocuit cu numele modulului `Main`, definit automat de `ghci` pentru fișierul tocmai încărcat.

```
*Main>
```

Modulele sunt unități elementare de structurare a codului despre care vom învăța în cursurile viitoare. Puteți reveni în **Prelude** folosind `:m - Main`.

Încercați să calculați `double myInt` din nou:

```
*Main> double myInt
```

Executați `double` cu alte argumente:

```
*Main> double 2000
```

Adăugați o funcție `triple` fișierului `lab1.hs`. Dacă fișierul este deja încărcat, puteți să îl reîncărcați folosind comanda ***reload*** (:r). Testați funcția `triple` pentru inputul `myInt`.

```
*Main> :r
Ok, 1 module loaded.
*Main> triple myInt
```

Elemente de limbaj

Există numeroase biblioteci utile de Haskell. Puteți găsi informații despre ele în **Hoogle**: https://hoogle.haskell.org/

Căutați în **Hoogle** funcția `head` folosită anterior. Observați că se găsește în mai multe biblioteci, printre care `Prelude` și `Data.List`.

3. Să presupunem că vrem să generăm toate permutările unei liste. Căutați în **Hoogle** folosind cuvântul-cheie `permutation` (sau ceva asemănător).

Printre rezultatele întoarse, se află și funcția `permutations` din biblioteca `Data.List`. Dați click pe numele funcției (sau al bibliotecii) pentru a citi mai multe detalii. Pentru a folosi funcția în interpretor, va trebui sa încărcați biblioteca `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 biblioteca folosind

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

Bibliotecile se includ în fișiere sursă folosind comanda `import`. Deschideți fișierul `lab1.hs` și adăugați linia de mai jos la începutul său:

```
import Data.List
```

Încărcați fișierul în interpretor și evaluați:

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

Ce se întâmplă? Hint: `[1..myInt]` este lista `[1,2,3,..., myInt]` care are multe elemente. (Întrebare bonus: de câte caractere este nevoie pentru a afișa toate elementele listei?)

Putem opri evaluarea unei expresii folosind `Ctrl+C`.

4. Căutați funcția `subsequences` în biblioteca `Data.List`, înțelegeți ce face și testați-o folosind câteva exemple.

Indentare

În Haskell se recomandă ***indentarea*** riguroasă a codului sursă. În anumite situații, nerespectarea regulilor de indentare poate provoca erori la încărcarea programului.

5. Modificați indentarea funcției `double` din fișierul `lab1.hs`. De exemplu:

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

Reîncărcați programul. Ce observați?

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

6. Definiți funcția `maxim`:

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

O variantă 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 variantă este:

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

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

O altă posibilitate ar fi să scriem funcția `maxim3` folosind expresii `let...in` astfel:

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

Atenție! Expresia `let...in` creează un domeniu de vizibilitate local.

O variantă cu indentări este:

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

Scrieți o funcție `maxim4` folosind `let..in` și indentări.

Scrieți o funcție care testează funcția `maxim4` prin care să verificați că rezultatul este mai mare (`>=`) decât fiecare din cele patru argumente. (hint: operatorii logici în Haskell sunt ||, &&, `not`).

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

Tipuri de date

7. 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]
```

`[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 in `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, iar funcția `maxim` are două argumente de tip `p` și întoarce un rezultat de tip `p`.

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

Exerciții

8. Scrieți următoarele funcții:
   a) o funcție cu doi parametri care calculează suma pătratelor lor;
   b) o funcție cu un parametru ce întoarce stringul "par" dacă parametrul este par și "impar" altfel;
   c) o funcție care calculează factorialul unui număr;
   d) o funcție care verifică dacă primul parametru este mai mare decât dublul celui de-al doilea parametru;
   e) o funcție care calculează elementul maxim al unei liste.

Citiți, citiți, citiți!

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