TP 1: prise en main et premiers codes

Plateforme

Pour programmer en utilisant le langage Haskell, il faut tout d'abord installer un compilateur. Nous allons utiliser le Glasgow Haskell Compiler qui est disponible ici : `https://www.haskell.org/ghc/`. Dans cette feuille on décrit l'utilisation élémentaire de ce compilateur sous (Ubuntu) Linux, donc les instructions ci-dessous sont valables pour les machines dans les salles UNIX de Dauphine et toute autre machine avec ce système d'exploitation.

Si vous voulez travailler dans votre propre machine vous avez plusieurs options. La solution qu'on vous propose est d'installer Ubuntu dans votre machine et puis installer Haskell avec la commande `sudo apt-get install ghc` (vous devez écrire cette commande sur une console/terminal). On remarque que, même si vous ne voulez/pouvez pas installer Ubuntu sur votre ordinateur, c'est toujours possible d'utiliser ce système d'exploitation à partir d'une clé USB [1]. Utiliser Ubuntu est une bonne idée, car cela vous garantit un environnement de travail identique à ce que vous trouverez à Dauphine. Plus généralement, se familiariser avec les systèmes de type Unix/Linux vous sera très utile.

Ceci dit, vous avez bien sûr aussi l'option d'utiliser un autre système d'exploitation (Windows/MacOS). L'utilisation de `ghc` est similaire sous ces systèmes. Vous pouvez suivre ce lien pour télécharger un compilateur de Haskell pour votre système : `https://www.haskell.org/platform/`.

Pour ceux qui n'ont jamais utilisé Linux

Si vous décidez d'utiliser Ubuntu, on vous fait quelques rappels. Pour ce cours, vous utiliserez deux outils:

• une *console* (ou *terminal*): c'est une fenêtre où vous pouvez taper des commandes. Vous pouvez ouvrir une console en cliquant sur le bouton se trouvant à gauche de l'écran. 

• un *éditeur de texte* qui vous servira à écrire votre code. Dans les salles `unix`, vous avez le choix d'utiliser `gedit` (éditeur par défaut) ou `GNU Emacs`. Pour ouvrir un éditeur, vous pouvez accéder au menu en utilisant la touche windows ou bien en cliquant sur le bouton  . Une fois dans le menu, cliquez sur l'icône de l'éditeur, si elle est absente, tapez le nom de l'éditeur.

Voici quelques commandes très utiles que vous pouvez utiliser dans un terminal:

- `cd` change directory, permet de changer de répertoire courant
  - `cd ..` permet d'accéder au répertoire père
  - `cd monRepertoire` permet d'accéder au répertoire `monRepertoire`
- `mkdir` make directory , permet de créer un répertoire: `mkdir Haskell` créé un répertoire Haskell.
- `ls` list, permet de connaître la liste des fichiers et répertoires contenus dans le répertoire courant.
- `more nomFichier` permet d'afficher le contenu du fichier `nomFichier`

[1] `https://ubuntu.com/tutorials/try-ubuntu-before-you-install`

Premier programme

Version interactive

La façon la plus simple de se lancer à la programmation en Haskell est d'utiliser la version interactive de GHC. Dans un terminal on écrit :

```
$ ghci
```

Si le compilateur est bien installé cela doit donner le message

```
GHCi, version 8.0.2: http://www.haskell.org/ghc/  :? for help
Prelude>
```

Maintenant on peut écrire quelques lignes en Haskell. Par exemple :

```
Prelude> 2+3+5
10
Prelude> sqrt (3^2 + 4^2)
5.0
```

Fichiers Haskell

La version interactive de GHC fonctionne très bien si on a envie d'écrire un programme de quelques lignes, mais devient rapidement peu pratique pour des programmes plus compliqués. On a donc besoin de stocker nos programmes dans des fichiers. On utilise par convention l'extension `.hs` pour les fichiers qui contiennent des programmes en Haskell. Pour s'habituer à l'utilisation de ce type de fichier, commencez avec le programme suivant :

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

Sauvegardez le programme ci-dessus dans un fichier nommé `fact.hs`. Puis, lancez `ghci` et écrivez :

```
Prelude> :load fact.hs
[1 of 1] Compiling Main             ( fact.hs, interpreted )
Ok, modules loaded: Main.
*Main> fact 5
120
```

L'instruction `:load` dit au compilateur de charger notre fichier. Cela nous permet d'utiliser toutes les fonctions et variables qui sont définies dedans. Il faut noter que si on fait une modification du fichier `fact.hs` le compilateur ne la prendra pas en compte. Par exemple, ajoutez à la fin du fichier les lignes :

```
x = 5
y = "abc"
```

Si vous essayez d'utiliser ces variables vous aurez le message suivant :

```
*Main> y

<interactive>:3:1: error: Variable not in scope: y
```

Pour faire le compilateur prendre en compte la nouvelle version du fichier il faut donner la commande `:reload`. On écrit :

```
*Main> :reload
[1 of 1] Compiling Main             ( fact.hs, interpreted )
Ok, modules loaded: Main.
*Main> y
"abc"
```

Vous êtes maintenant prêts à commencer à programmer en Haskell ! Le mode d'utilisation habituel est qu'on garde un fichier ouvert sur lequel on travaille, et on recharge périodiquement notre programme sur `ghci`. On note que les commandes `:load` et `:reload` peuvent être raccourcies en `:l` et `:r`. Si votre fichier contient une erreur, le compilateur va vous avertir quand vous essayez de le charger avec une indication de la ligne problématique. Vous devrez donc corriger votre programme avant de le charger.

Programmer des solutions pour les problèmes suivant et, une fois finis, montrez vos programmes à votre chargé de TP.

Exercices

NB: pour toutes vos fonctions vous devez bien préciser leurs types ! Pour ce TP vous devez programmer vos fonctions en utilisant seulement des opérations élémentaires sur les listes (p.ex. `head`, `tail` etc.). N'utilisez pas de fonctions d'ordre supérieure (p.ex. `map`), ni l'opération `<-`.

1 Plus grand commun diviseur

Écrire une fonction `pgcd` qui prend en argument deux entiers non-négatifs et retourne leur plus grand commun diviseur. **NB:** Pour rappel, l'opérateur `/` fait une division de nombres réels en Haskell. La division des nombres entiers est effectuée en utilisant les opérateurs `div` et `mod`. On écrit par exemple

```
*Main> 15 `div` 2
7
*Main> div 15 2
7
*Main> 15 `mod` 11
4
```

2 Listes Triées

Écrire une fonction `isSorted` qui prend en argument une liste de nombres entiers et retourne `True` si la liste est triée en ordre strictement croissant.

3 Nombre premier

Implémenter une fonction `isPrime` qui prend en argument un entier et qui retourne un booléen indiquant si l'entier est un nombre premier ou non.

4 Matrices Transposées

Écrire une fonction qui prend en argument une matrice (c'est-à-dire une liste de listes de nombres entiers) et retourne la matrice transposée. Par exemple, étant donné `[[1,2,3],[4,5,6]]` votre fonction doit retourner `[[1,4],[2,5],[3,6]]`.

5 Matrices

Cet exercice fait suite à l'exercice 4 du TP1. Pour rappel, on utilise les listes de listes pour représenter des matrices.

1. Écrire une fonction `isMatrix` qui retourne `True` si la liste passée en argument est bien une matrice. Rappel : Dans une matrice toutes les lignes doivent avoir la même taille. Écrire aussi deux fonctions `rows` et `columns` qui retournent le nombre de lignes et colonnes respectivement d'une matrice donnée en paramètre.
2. Écrire une fonction `addMatrix` qui, étant donné deux matrices A, B retourne $A + B$ (sous la condition que les deux matrices ont les mêmes dimensions).
3. Écrire une fonction `multMatrix` qui retourne le produit de deux matrices A, B passées en paramètre (sous la condition que le nombre de colonnes de A est égale au nombre de lignes de B).

Exemple :

```
*Main> multMatrix [[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]]
[[22,28],[49,64]]
```

6 La conjecture de Collatz

On considère la fonction suivante :

$$f(x) = \begin{cases} 3x+1 & \text{si } x \text{ est impaire} \\ x/2 & \text{sinon} \end{cases}$$

La conjecture de Collatz (ou conjecture de Syracuse) est un problème ouvert de mathématiques qui dit qu'en partant de n'importe quel entier positif x, si on applique la fonction f de façon exhaustive, on va toujours arriver à 1.

Écrire une fonction qui, étant donné un entier positif x retourne la suite de Collatz qu'on obtient si on applique la fonction f jusqu'à ce qu'on arrive à 1. Par exemple :

```
*Main> collatz 23
[23,70,35,106,53,160,80,40,20,10,5,16,8,4,2,1]
```