PF TP1 - Découverte d'Haskell

2023-08-21

Contents

Environnement de développement

Mise en place des outils

- Allez dans le dossier `ulco-pf-etudiant/tp1` et ouvrez le dossier avec VSCode :

```
cd ulco-pf-etudiant/tp1

code .
```

Découverte des outils Haskell

- Ouvrez le fichier `hello.hs`. Vérifiez que vous avez bien la coloration syntaxique.
- Exécutez votre fichier avec `runghc`. Vérifiez que cela ne produit pas de fichier intermédiaire.

- Exécutez votre fichier en activant les warnings. Vérifiez qu'un warning indique la signature de fonction manquante. Ajoutez cette signature et vérifiez qu'il n'y a plus de warning.
- Compilez avec `ghc` et exécutez le programme. Vérifiez les fichiers intermédiaires produits puis supprimez-les.
- Lancez l'interpréteur avec `ghci`, chargez votre fichier et exécutez le programme.
- Dans un navigateur web, connectez-vous à Hoogle. Cherchez la signature de la fonction `putStrLn`. Puis cherchez toutes les fonctions de signature `String -> IO ()`.
- Dans un navigateur web, connectez-vous à Hackage, cherchez la bibliothèque `base` puis le module `Prelude` puis la fonction `putStrLn`.

Expressions

Exécutez ces expressions dans ghci

1. `21*2`
2. `22+10*2`
3. `1 == 2`
4. `1 /= 2`
5. `2+(-2)`
6. `div 21 2`
7. `21 `div` 2`
8. `(*) 21 2`
9. `42 `mod` 2 == 0`
10. `even 42`

Idem

1. `y = 84 `div` 2`
2. `y`
3. `40 + if even 20 then 2 else 3`
4. `'Z' < 'a'`
5. `"abc" <= "ab"`
6. `"abc" >= "ac"`
7. `1 + 2 * 3`
8. `5.0 - 4.2 / 2.1`
9. `3 > 4 || 5 < 6 && not (7 /= 8)`

Système de types

Affichez le type de ces expressions

1. `'Z'`
2. `'Z' < 'A'`
3. `"Z"`
4. `2.1`
5. `2.1 :: Double`
6. `2`
7. `2 :: Int`
8. `if True then 2 else 2.1`
9. `21 `div` 2`
10. `21 / 2`
11. `fromIntegral (21 `div` 2) / 3`

Fonctions

Premières fonctions

- Exécutez les expressions suivantes dans `ghci`.

  1. `f x = 2*x + 1`
  2. `f 3`
  3. `f 4`
  4. `g x = 2.1 * f x`
  5. `g 2`
  6. `g 1`

- Quel est le type de `f` et celui de `g` ?

Fonctions à plusieurs paramètres

- Exécutez les expressions suivantes :

  1. `h x y = 2*x - y`
  2. `h 2 1`
  3. `h 2 1.2`

- Quel est le type de chacune des expressions suivantes :

  1. `h`
  2. `h 2 1`
  3. `h 2 1.2`
  4. `h 2`
  5. `h 2.1`

Typage de fonctions

- Dans un fichier `add.hs`, écrivez une fonction `add2Int` qui prend deux **entiers** et retourne leur somme. Testez (avec `ghci` ou dans un `main`) sur les valeurs `2 3` et sur `2 3.4`. Que se passe-t-il ?
- Écrivez une fonction `add2Double` identique mais qui prend deux **doubles**. Testez sur les mêmes valeurs. Que se passe-t-il ?
- Écrivez une fonction `add2Num` identique mais fonctionnant sur le type le plus générique possible. Testez sur les mêmes valeurs.
- Définissez un **entier** `u` de valeur `2` et un **double** `v` de valeur `3`. Exécutez `add2Num` sur les valeurs `u u`, `v v` et `u v`. Que se passe-t-il ?

Entrées/sorties

Affichages et saisies

- Dans un fichier `io.hs`, écrivez un programme qui affiche un texte, saisit une ligne puis affiche le texte saisi.

```
$ runghc io.hs
Entrez le texte : foo bar
Vous avez entré : foo bar
```

Arguments de la ligne de commande

- Dans un fichier `args.hs`, écrivez un programme qui affiche les arguments de la ligne de commande.

```
$ runghc args.hs foo bar
les arguments sont : ["foo","bar"]
```

Nombres aléatoires

- Dans un fichier `random.hs`, écrivez un programme qui affiche un nombre aléatoire entre 0 et 100. Indication : utilisez la fonction randomRIO.

```
$ runghc random.hs
4
$ runghc random.hs
16
$ runghc random.hs
86
```

Récapitulatif

Hello-goodbye

- Écrivez un programme qui, en boucle, demande de saisir un nom et affiche un message d'accueil. Le programme termine quand on saisit un texte vide. Indication : la fonction `main` peut être récursive.

```
$ runghc hello-goodbye.hs
What's your name: Toto
Hello Toto!
What's your name: Julien
Hello Julien!
What's your name:
Goodbye!
```

Guessing Game

- Écrivez un programme qui choisit un nombre aléatoirement puis demande à l'utilisateur de le deviner en 10 tentatives ou moins. On suppose que l'utilisateur saisit bien des nombres.

```
$ runghc guessing-game.hs
Type a number (10 tries): 50
Too big!
Type a number (9 tries): 25
Too big!
Type a number (8 tries): 12
You win!
```

```
$ runghc guessing-game.hs
Type a number (10 tries): 2
Too small!
...
Type a number (1 tries): 2
Too small!
You lose! Was 89
```

Fibonacci

- Écrivez un programme qui affiche le $n^{\text{ième}}$ terme de la suite de Fibonacci où n est passé en argument de ligne de commande. Pour l'instant, implémenter naïvement la définition : $F(n) = F(n-1) + F(n-2)$.

```
$ ghc -O2 fibo.hs

$ ./fibo
usage: <n>

$ ./fibo 10
f(10) = 55

$ ./fibo 11
f(11) = 89

$ ./fibo 9
f(9) = 34
```

- Réécrivez la fonction `fibo` avec une complexité linéaire selon n et qui fonctionne pour des grands nombres.

```
$ ghc -O2 fibo.hs

$ time ./fibo 10000
f(10000) = 33644764876431783266621612005107543310 3...

real    0m0,017s
user    0m0,003s
sys     0m0,003s
```