UNIVERSITÉ CAEN NORMANDIE

Département mathématiques et informatique
UFR des Sciences

Programmation Fonctionnelle
Licence 2 Informatique
L2 MATH
A. Ouali

TP1 : Découverte d'Haskell

"Premiers pas : `GHC`, expressions, types élémentaires, fonctions, listes et tuplets"

1 Environnement de développement

1. Pour ce TP, travaillez dans le dossier ∼/Documents/L2/PF/tp1/. "∼" représente votre dossier personnel.
2. En utilisant un éditeur de texte (gedit, nano, vim, etc.), écrivez un fichier **tp1.hs** qui affiche le texte "hello" (pour l'instant, n'écrivez pas la signature de fonction, voir le cours).
3. Ouvrez un terminal en vous mettant dans le même répertoire. En préférence, placez l'éditeur de texte à côté du terminal pour avoir une ergonomie plus confortable.
4. Mode compilation :
   — Exécutez votre fichier avec `runghc`. Vérifiez que cela ne produit pas de fichier intermédiaire dans votre répertoire.
   — 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.
5. Mode interactif :
   — Lancez l'interpréteur Haskell avec `ghci`, chargez votre fichier et exécutez le programme.

2 Quelques expressions à étudier

Exécutez les expressions suivantes dans `ghci` et expliquez en commentaires dans votre fichier chacune d'elles :

1. 20 * 3
2. (*) 20 3
3. 22+10*2
4. 1 == 2
5. 2 + (-2)
6. [1..10]
7. [1.0, 1.25..2.0]
8. [1, 4..15]
9. [1.0..1.8]
10. 'F' < 'P'
11. "FPhaskell" < "FPH"
12. div 1 2
13. 21 / 2
14. 4 `div` 2
15. 3 > 4 || 5 < 6 && not (7 /= 8)
16. 10 + if even 20 then 2 else 3

3 Fonctions

Exécutez les expressions suivantes dans `ghci` :
— f x = 2 * x + 1
— g x = 2.1 * f x
— h x y = 2*x - y

1. A quoi sert les trois expressions ? Donnez le nombre d'arguments dans chacune.
2. Donnez l'expression pour appliquer chaque fonction sur des valeurs positives et négatives de votre choix.
3. Quel est le type de f, de g et celui de h ? récrivez ces fonctions dans votre `tp1.hs` en précisant la signature de chaque fonction.

3.1 Syntaxe

Le script ci-dessous contient des erreurs de syntaxe :

```
N = a `div` length xs
    where
        a = 10
      xs = [1 ,2 ,3 ,4 ,5]
```

1. trouvez ces erreurs en les expliquant en commentaires dans votre tp1.hs,
2. corrigez-les, de façon à pouvoir compiler le code et le tester avec ghci.

4 Utilisation des types de base dans les fonctions

4.1 Type de base `Int`

On considère des nombres entiers de type `Int`.
En précisant la signature des fonctions, définissez :
— la fonction `square` qui élève un nombre au carré,
— la fonction `sumSquares` qui calcule la somme des carrés de deux nombres,
— la fonction `sumSquaresMax` qui, à trois nombres, associe la somme des carrés des deux plus grands. Exemples :

```
> sumSquaresMax 5 6 7  ==> 85
> sumSquaresMax 7 6 5  ==> 85
> sumSquaresMax 6 7 5  ==> 85
```

4.2 Type de base `Float`

1. Conversions entre températures exprimées en degrés Celsius et en degrés Fahrenheit. Le lien entre les deux est donné par l'équation : $C = 5/9(F - 32)$. Définissez la fonction réciproque `f2c` en précisant sa signature.
2. Définir une fonction qui prend comme paramètres un nombre de kilomètres et un nombre de personnes, et retourne le montant à rembourser, sachant que :
   — le tarif appliqué est de 0.52 F du km,
   — une réduction de 25 % est appliquée sur le prix au km entre 2 et 4 personnes,
   — une réduction de 50 % est appliquée sur le prix au km entre 5 et 10 personnes,
   — une réduction de 75 % est appliquée sur le prix au km à partir de 11 personnes.

```
> travelExpenses 150 3   ==> 175.5
> travelExpenses 100 7   ==> 182.0
> travelExpenses 90 3  ==> 105.299995
```

Indication. On pourra utiliser une fonction qui revoit la réduction en fonction du nombre de personnes. Cette fonction pourra être utilisée par la fonction qui calcule le montant à rembourser.

4.3 Type de base `Char`

Définir la fonction `(nextUpperCase c)` qui prend une lettre majuscule et retourne la lettre majuscule suivante ; on fera en sorte que `'A'` suive immédiatement `'Z'`.

```
> nextUpperCase 'V'  ==> 'W'
> nextUpperCase 'Z'  ==> 'A'
```

Indication. On pourra utiliser les deux fonctions suivantes pour coder/décoder chaque lettre :

```
decode :: Int -> Char          code :: Char -> Int
decode n = toEnum n            code c = fromEnum c
```

5 Triples de Pythagore

Un triplet de Pythagore[1] est défini par un ensemble de trois entiers (a, b, c) qui vérifient l'équation $a^2 + b^2 = c^2$. Par exemple, $(3, 4, 5)$ est un triplet de Pythagore, puisque $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

Nous utilisons dans cet exercice une approche plus moderne pour ce problème très ancien en utilisant Haskell pour générer et vérifier des triplets de Pythagore.

1. Définir une fonction `isTriple` qui teste les triplets de Pythagore. Ne pas se soucier des triplets avec des côtés de longueur négative ou nulle. Préciser le type de la fonction (signature) avant sa définition. Tester la fonction dans `ghci` :

   ```
   Main> isTriple 3 4 5
   True
   Main> isTriple 3 4 6
   False
   ```

2. On veut créer des triplets automatiquement. Une formule simple pour trouver des triplets de Pythagore est la suivante : $(x^2 - y^2, 2\ y\ x, x^2 + y^2)$ est un triplet de Pythagore pour tous les entiers positifs x et y avec $x > y$. Ces dernières conditions sont pour garantir que les côtés du triangle sont positifs ; pour cet exercice, nous négligeons cela.
   Définir les fonctions `leg1`, `leg2` et `hyp` avec leur type pour géner les composants des triplets de Pythagore en utilisant les formules ci-dessus.

   ```
   Main>leg1 5 4
   9
   Main>leg2 5 4
   40
   Main>hyp 5 4
   41
   Main>isTriple 9 40 41
   True
   ```

3. Définir une fonction `Pythest` pour tester facilement sur différentes valeurs de x et y.

6 Nombres complexes avec des tuples

Un nombre complexe peut être représenté avec un couple de nombres réels avec comme premier élément la partie réelle et l'autre élément est la partie imaginaire.

1. Pythagore était un philosophe grec qui a vécu entre environ 580 et 490 av. J.-C. Il est connu comme la personne qui a découvert la relation entre les côtés d'un triangle et un triangle à angle-droit.

1. En utilisant cette représentation, écrivez un code Haskell pour implémenter des opérations élémentaires sur les nombres complexes, notamment la somme, le produit et la division. Ajoutez dans votre code une déclaration du type de la fonction (signature) pour chaque fonction.
2. Chargez votre code dans `ghci`, et calculez les expressions suivantes :
   — $(1+i)(2+3i)$
   — $i(1+i)+1-i$
   — $\sum_{n=1}^{50} ni$

7 Tous pairs

Définir récursivement la fonction `(allEven xs)` qui détermine si la liste `xs` est uniquement composée de nombres pairs. Ajoutez la déclaration du type de la fonction avant sa définition.
Exemples :

```
> allEven [2,4,6,8,10]  ==> True
> allEven [2,3,6,8]  ==> False
```