14/01/2013
Durée : 2 heures
Documents autorisés

Programmation fonctionnelle
Master 1 Informatique

Attention : donnez autant de détails que vous jugez nécessaire ; des simples copies-collers-miroirs de vos notes de cours ne seront pas jugés satisfaisants. En bref, montrez que vous avez compris.

Types

Exercice 1. Calculez le type et, éventuellement, les contraintes de classe du type, de chacune des expressions et fonctions suivantes :

```
tailleAlphabet = ord 'z' - ord 'A'
encoder [] = 0
encoder (x:xs) = (ord x) - (ord 'A') + tailleAlphabet * (encoder xs)
decoder 0 = []
decoder x =
    (chr (x `mod` tailleAlphabet + ord 'A')):(decoder (x `div` tailleAlphabet))

absMax [] = -1
absMax (x:xs) = max (abs x) (absMax xs)

fG f g [] = -1
fG f g (x:xs) = g (f x) (fG f g xs)
```

Rappel :

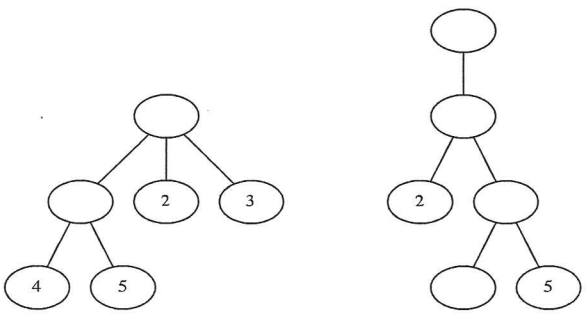
```
ord :: Char -> Int -- Transforme un caractere dans son code ascii
chr :: Int -> Char -- Transforme un entier dans le caractere correspondant
                   -- (selon ascii)
```

Types récursifs

Pour les exercices suivants, considérez le type (récursif) des arbres dont chaque noeud possède une liste ordonnée d'enfants, et dont chaque feuille est étiquetée par un entier. Le code Haskell qui définit ce type est le suivant :

```
data Arbre = Feuille Int | Noeud [Arbre]
```

Exercice 2. Écrivez deux expressions qui codent en Haskell ces deux arbres :



Exercice 3. Écrivez un script Haskell contenant la définition de :
– une fonction `maximum`, qui calcule l'étiquette maximum d'une feuille de l'arbre passé en paramètre,

- une fonction `somme`, qui calcule la somme de toutes les étiquettes des feuilles de l'arbre passé en paramètre,
- une fonction `profondeur`, qui calcule la longueur d'une branche de longueur de maximale de l'arbre en paramètre.

Exercice 4.
- Complétez le script de l'exercice précédent en y définissant une fonction

```
foldArbre :: (Int -> a) -> ([a] -> a) -> Arbre -> a
```

  telle que :
  - `foldArbre f g` appliquée à une feuille, produira la valeur de `f` sur l'étiquette,
  - `foldArbre f g` appliquée à un noeud produira la valeur de `g` appliquée à la liste obtenue des appels récursifs sur les enfants.

  Par exemple, `foldArbre (+1) sum`, appliqué à l'arbre sur la gauche de l'exercice 2, produira `18` comme résultat.
- Montrez comment les fonctions `maximum`, `somme` et `profondeur` (de l'exercice précédent) peuvent se définir à partir de la fonction `foldArbre`.

Évaluation

Exercice 5. Évaluez, par nom (stratégie call-by-name) *et* par valeur (stratégie call-by-value), les expressions `e1`, `e2` et `e3`, du script Haskell suivant :

```
e1 = (\x y -> x + y) (1 + 2) (3 + 4)
omega = omega + 1
e2 = (\x y -> y) omega 33
ifthenelse x y z = if x then y else z
e3 = ifthenelse True 0 omega
```

Polynômes (divertissement)

Exercice 6. Écrivez un script Haskell qui contiendra :
- la définition d'un type `Pol`, pour représenter les polynômes à coefficients entiers,
- une fonction `eval :: Pol -> Int -> Int`, qui évalue un polynôme en un entiere donné,
- une fonction `toPol :: Int -> Int -> Pol` telle que, appliquée à deux entier n et m, produira le polynôme $P(x) = \sum_i a_i x^i$ tel que $P(m) = n$.

Dans cet exercice, on vous demande de faire preuve de votre maîtrise du langage Haskell : des erreurs de syntaxe ou de typage seront tenu en compte pour la correction.