Travaux Pratiques d'AP3 n°4

—Licence Informatique, 3ième année, 1ier semestre—

▸ **Exercice 1.** *L'empilement des appels récursifs pour une fonction récursive non terminale peut poser des problèmes de gestion de la mémoire.*

1. *Écrivez une fonction récursive non terminale* `create_list(n : int) : int list` *qui crée la liste des n premiers entiers. Lancez son exécution pour* `create_list(100000000)` *et observez ce qu'il se passe. Donnez une explication.*
2. *En introduisant une fonction* `create_list_rt_aux (n, l : int * int list) : int list` *récursive terminale, écrivez une fonction* `create_list_rt(n : int) : int list` *qui crée la liste des n premiers entiers. Lancez son exécution pour* `create_list_rt(100000000)` *et observez ce qu'il se passe. Donnez une explication.*
3. *Réécrivez la fonction* `create_list_rt(n : int) : int list` *en utilisant la fonction* `whileloop (r, cont, suiv : 'a * ('a -> bool)* ('a -> 'a)) : 'a` *vue en cours et rappelée ci-dessous.*

   ```
   let rec whileloop (r, cont, suiv : 'a * ('a -> bool) * ('a -> 'a)) : 'a =
     if not(cont(r)) then r else whileloop(suiv(r), cont, suiv)
   ;;
   ```

   Lancez l'exécution de `create_list_rt(100000000)` *et observez ce qu'il se passe. Que pouvez-vous conclure ?*
4. *Transformez la fonction précédente en une fonction* `create_list_rt_lg(n : int) : int list` *qui utilise la boucle* `while` *de OCaml. Expliquez la transformation que vous avez réalisée (dites où se retrouvent les différents éléments de la fonction* `whileloop`*).*
5. *Dans le module* `List` *sur les listes, il y a une fonction* `length`*. Lancez l'exécution de* `List.length(create_list_rt_lg(100000000))` *que pouvez-vous conclure sur la programmation de la fonction* `length` *? Vérifiez-le en écrivant une version non récursive terminale* `longueur` *qui calcule la longueur d'une liste. (Vous pouvez refaire les points 3 et 4 avec la fonction* `longueur`*).*

▸ **Exercice 2.** *La fonction qui code directement à partir de sa définition la suite de Fibonacci est un exemple bien connu de fonction doublement récursive*[(1)]*.*

(1). Vous connaissez aussi sans doute une version simplement récursive du calcul des termes de la suite de Fibonacci qui utilise deux termes consécutifs.

1. *La suite de Fibonacci étant définie par* $F_0 = 0$, $F_1 = 1$ *et* $F_n = F_{n-1} + F_{n-2}$ *pour* $N > 2$, *écriver une fonction* `fibo` *(doublement récursive) qui calcule un terme de la suite de Fibonacci.*
   (a) *Estimez la complexité de cette fonction en prenant comme opération fondamentale le nombre d'appels récursifs.*
   (b) *Essayez votre fonction pour* $n = 35$.
2. *En vous aidant de la version itérative de la fonction de parcours infixe d'un arbre binaire vue en cours, donnez une version itérative* `fibo_iter` *de la fonction* `fibo`.
   (a) *Essayez votre fonction pour* $n = 35$. *Que constatez-vous par rapport à l'essai de la fonction précédente ?*
   (b) *Expliquez la différence constatée.*