TD 4 - Récursivité (fonctions récursives, listes)

Exercice 1. Ecrire les fonctions récursives suivantes.

• `fact` : factorielle d'un entier naturel `n`. (Que fera la fonction pour n=−1?)
• `gfact` : factorielle généralisée à tout entier relatif.
• Fibonacci d'un entier naturel : `fibo(1)=fibo(0)=1` ; `fibo(n>1)=fibo(n-1)+fibo(n-2)`. N.B. : la fonction ne devra avoir qu'un seul appel rÃ©cursif terminal.
• Fibonacci généralisée à tout réel : on prend la partie entière du réel et la fonction retourne 1 pour tout réel négatif.
• Fonction qui calcule, sans utiliser de multiplication, le n-ième terme d'une suite arithmétique de premier terme $U_0 = x$ et de raison P.
• Fonction qui calcule, sans utiliser la fonction puissance, le n-ième terme d'une suite géométrique de premier terme $V_0 = x$ et de raison R.
• Fonction qui retourne le nombre de chiffres d'un entier passé en paramètre.
• Fonction qui retourne la somme des chiffres d'un entier passé en paramètre.
• Fonction qui calcule la somme des chiffres d'un entier passé en paramètre, additionne les chiffres de cette somme, et ainsi de suite jusqu'à obtenir un nombre à un chiffre.
• Fonction `puissance` qui retourne l'entier n^p avec n entier relatif et p entier naturel.

Exercice 2. Pour écrire les fonctions récursives suivantes, on pourra utiliser les fonctions `String.make n c` qui construit une chaîne de `n` caractères `c` (par exemple, `String.make 3 'a'` retourne `"aaa"`) et `String.sub s d n` qui extrait de `s` une sous-chaîne de longueur `n` qui commence à l'indice `d` de `s` (`String.sub "abcdef" 1 3` retourne `"bcd"`).

• Fonction `stringofchar` qui transforme un caractère en une chaîne.
• Fonction `palindrome` qui vérifie si un mot passé en paramètre est un palindrome.
Définition de Wikipedia : Un palindrome est une figure de style désignant un texte ou un mot dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche, comme dans le mot "ici" ou l'expression "esoperesteicietserepose".
• Fonction inversant les lettres d'une chaîne de caractères (par exp. "abc" donnera "cba").
• Fonction qui encode une chaîne de caractères avec le code de Jules César de rang n∈ [1..25] (par exemple avec n=3: 'a'→'d', 'b'→'e', ..., 'z'→'c'). On pourra utiliser une fonction annexe qui code un caractère.

Exercice 3. On veut écrire deux fonctions `estpair` qui détermine si un nombre est pair, et `estimpair` qui détermine si un entier est impair ; ces deux fonctions doivent être mutuellement récursives, c'est-à-dire que chacune doit appeler l'autre. Quelle sera la difficulté ?

Exercice 4. Créer les fonctions suivantes :

• `listofstring` qui transforme une chaîne de caractères en une liste de caractères.

• `stringoflist` qui transforme une liste de caractères en chaîne de caractères.

• `souschaine s d t` qui extrait de la chaîne `s` une sous-chaîne de longueur `t` commençant à l'indice `d`. (sans utiliser `String.sub`).

• `contient s c` qui vérifie si le caractère `c` appartient à la chaîne `s`
(sans utiliser `String.contains`).

Exercice 5. En utilisant autant que de besoin les fonctions prédéfinies `List.hd`, `List.tl`, `List.rev`, et les opérateurs :: et @, et pour `s=[1;2;5;6;8]` et `t=[[1;2;5];[6;7]]`, quelle commande permet de :

• Accéder au deuxième élément de s ? au quatrième ? Supprimer les deux premiers éléments de s ?

• Accéder à l'élément de valeur 7 dans t ?

• Transformer t en [1;2;5;6;7] ?

• Transformer t en [[7;6];[5;2;1]] ?

Exercice 6. Dans cette exercice, on n'utilisera pas les fonctions de liste prédéfinies (`List.hd` : tête d'une liste non vide, `List.tl` : queue d'une liste non vide, `List.rev` : inversion, `List.length` : taillee, `List.nth` : n-ième élément, etc.) On pourra par contre utiliser les opérateurs `@` (concaténation de listes) – sauf pour `fusion` – et `::` (ajout en tête).

Créer les fonctions* curryfiées de listes :

* car, cdr, cons, et cadr sont des fonctions prédéfinies du langage fonctionnel LISP.

- `car`, qui retourne la tête d'une liste non vide.
- `cdr`, qui retourne la queue d'une liste.
- `cons`, qui ajoute un élément en tête d'une liste.
- `cadr`, qui retourne le deuxième élément d'une liste non vide.
- `estvide`, qui teste si une liste est vide.
- `longueur1`, qui retourne la taille de la liste (càd son nombre d'éléments), en utilisant un match.
- `longueur2`, qui retourne la taille de la liste sans utiliser de match.
- `memetaille`, qui teste si deux listes ont la même taille (faire une version avec if, une avec match).
- `fusion`, qui concatène deux listes, sans utiliser `@`, mais en utilisant si besoin `car`, `cdr` et `cons`.
- `inverser`, qui inverse une liste, en utilisant si besoin `car`, `cdr` et `cons`.

Exercice 7. Sans utiliser de fonctions d'ordre supérieur*, Créer les fonctions suivantes.

**Une fonction d'ordre supérieure prend une/des fonctions en paramètre.*

- `debut`, qui retourne les `k` premiers éléments d'une liste `s`.
- `fin`, qui retourne les `k` derniers éléments d'une liste `s`.
- `sommeliste`, qui fait la somme des éléments d'une liste d'entiers.
- `produitliste`, qui fait le produit des éléments d'une liste de réels.
- `appartient`, qui teste si un élément `x` est présent dans une liste `s`. Est-ce un problème s'il y a plusieurs occurrences d'une même valeur dans la liste ?
- `oter`, qui retire la première occurrence d'une valeur `x` dans une liste `s`. (la liste est inchangée si elle ne contient aucune occurrence de x.)
- `eliminer`, qui retire toutes les occurrences d'une valeur `x` dans une liste `s`, s'il s'y trouve. (la liste est inchangée si elle ne contient aucune occurrence de x.)
- `expurger`, qui ne conserve qu'un unique exemplaire de chaque élément d'une liste `s`.

Exercice 8. Ecrire une fonction récursive qui, appliquée à une fonction f, calcule $\sum_{n=0}^{p} f(n)$

Exercice 9. Ecrire les fonctions récursives suivantes.

- Fonction qui calcule la somme des carrés des éléments d'une liste d'entiers. De quel type sera votre fonction ?
- Fonction qui détermine la valeur maximale d'une liste d'entiers. On pourra utiliser `min_int` et `max_int`.
- Fonction qui calcule la liste des sommes terme à terme de deux listes d'entiers. Que fait-on si les listes n'ont pas la même longueur ?

Exercice 10. Ecrire une fonction qui retourne la suite de Syracuse d'un entier n.
Que se passe-t-il si on saisit un nombre négatif ? Si la conjecture était fausse pour un nombre donné ? Si on oubliait de mettre la condition de fin quand on arrive à 1 ?

Définition de Wikipedia : On part d'un nombre entier plus grand que zéro ; s'il est pair, on le divise par 2, s'il est impair, on le multiplie par 3 et on ajoute 1. En répétant l'opération jusqu'à atteindre 1, on obtient une suite d'entiers positifs dont chacun ne dépend que de son prédécesseur. Par exemple, à partir de 14, on construit la suite des nombres : 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. C'est ce qu'on appelle la suite de Syracuse du nombre 14.
Une conjecture affirme que toute suite de Syracuse est finie : on arrive toujours à 1.