TP n°8 - Récursivité, listes OCaml.

Tous les codes de ce TP devront être remis sur Moodle - TP8 - avant jeudi 24/11 23h59

Exercice 1 (Travail sur les listes OCaml - Échauffement).
Tous les fonctions ci-dessous seront écrites en OCaml dans un fichier `exercices-listes.ml`.
On essaiera au maximum d'utiliser le paradigme fonctionnel pour coder ces fonctions, en utilisant au maximum les syntaxes idiomatiques du langage OCaml.
On veillera également à appliquer les principes de la programmation défensive et à tester abondamment et intelligemment les fonctions (notamment en essayant de couvrir toutes les alternatives pouvant survenir).
L'utilisation du module `List` est interdite dans cet exercice.
On réfléchira sur papier, et on s'attachera à bien définir les entrées et les sorties avant de se lancer dans la programmation.
Toutes les fonctions devront être récursives.

1. Écrire une fonction `longueur` qui retourne la taille d'une liste
2. Écrire une fonction `miroir` qui retourne la liste miroir d'une liste
3. Écrire une fonction `dernier` qui retourne le dernier élément d'une liste
4. Écrire une fonction `insere_fin` qui insère une nouvelle valeur à la fin d'une liste
5. Écrire une fonction `supprime` qui supprime **toutes les occurrences** d'une valeur si elle est présente dans une liste
6. Écrire une fonction `concatene` qui concatène deux listes
7. Écrire une fonction `double` qui dédouble tous les éléments d'une liste : la liste renvoyée est de taille deux fois plus grande que celle en entrée, et tous les éléments de la liste de départ y apparaissent deux fois consécutivement
8. Écrire une fonction `n_ieme` qui retourne le n-ième élément d'une liste

Exercice 2 (Compteur de voyelles).
On reprend l'exercice d'implémentation du compteur de voyelles. Reprenez le code `double_voyelles.c` implémenté dans le TP5, copiez le dans le TP6 en le renommant `voyelles.c`.
Implémentez en C dans ce fichier une fonction **itérative** `compteur_voyelles` qui compte le nombre de voyelles d'une chaîne de caractères. Nettoyer le code, commentez ou effacer les lignes de codes issues de l'ancien TP qui ne resserviront pas.
Testez votre fonction.
Implémentez en C dans ce fichier une fonction **récursive** `compteur_voyelles_rec` qui fait le même travail. Testez la !

Exercice 3 (Évolution régulière d'une quantité).
Des chercheurs étudient une population de chevreuils dans une zone forestière. La population de ces animaux augmente de 3% par an depuis 1995. La population totale de ces animaux était de 2500 en 1999. Dans un fichier `evolution.ml` :

1. Quelle suite permet de modéliser l'évolution de cette population d'animaux ?
2. Implémenter une fonction `récursive nombre_animaux` permettant de calculer le nombre d'animaux pour n'importe quelle année à partir de 1995, en faisant l'hypothèse que cette évolution se poursuive de manière identique dans les années futures. Tester.
3. Implémenter une fonction **itérative** `annee_nombre_animaux_superieur_a` qui renvoie l'année à partir de laquelle cette population dépassera un seuil donné. Tester.
4. (A réfléchir à la maison) Nous allons améliorer votre code pour le rendre générique. Nous allons créer une fonction générique `evolution_constante` donnant la valeur d'une quantité pour n'importe quelle année, connaissant cette quantité pour une certaine année de départ et son taux d'évolution, supposé constant. Écrire cette fonction d'évolution `evolution_constante` générique.
5. Tester la fonction `evolution_constante` dans le cas particulier de cette population de chevreuils.
6. Redéfinir la fonction `nombre_animaux` à partir de la fonction `evolution_constante`. *Indication : On pourra utiliser la curryfication et la notion d'application partielle.*
7. (Plus difficile, à faire à la maison) Écrire la fonction `annee_nombre_animaux_superieur_a` sous forme récursive

Exercice 4 (Travail sur les listes OCaml - Suite).
Tous les fonctions ci-dessous seront écrites en OCaml, toujours dans le fichier `exercices-listes.ml`.
L'utilisation du module `List` est interdite dans cet exercice.
On réfléchira sur papier, et on s'attachera à bien définir les entrées et les sorties avant de se lancer dans la programmation.
Cet exercice devra être fini à la maison.
Toutes les fonctions devront être récursives.

1. Écrire une fonction `insere` qui insère une nouvelle valeur à une position donnée dans une liste
2. Écrire une fonction `appartient` qui indique si une valeur est présente dans une liste
3. Écrire une fonction `supprime_doublons` qui supprime les doublons d'une liste. On pourra utiliser l'une des fonctions précédemment codées.
4. Écrire une fonction `map` qui fait le même travail que la fonction du même nom dans le module `List` [a]
5. Écrire une fonction `for_all` qui fait le même travail que la fonction du même nom dans le module `List`
6. Écrire une fonction `fold_left` qui fait le même travail que la fonction du même nom dans le module `List`
7. On appelle doublon une valeur qui est présente 2 fois ou plus dans une structure de données. Écrire une fonction `a_doublon` qui indique si une liste présente ou non des doublons
8. Écrire une fonction `maximum` qui renvoie la valeur maximale d'une liste
9. Écrire une fonction `minimum` qui renvoie la valeur maximale d'une liste

a. https://v2.ocaml.org/api/List.html