TD 5 (Fonctions d'ordre supérieur)

Exercice 1. Connaissant la définition des fonctions d'ordre supérieur de référence :

- `List.map` :
  prend en paramètres une fonction f et une liste $[x_1 ; x_2 ; \ldots ; x_n]$,
  et retourne la liste $[f(x_1) ; f(x_2) ; \ldots ; f(x_n)]$.
- `List.fold_right` :
  prend en paramètres une fonction f, une liste $[x_1 ; x_2 ; \ldots ; x_n]$ et un cas de base b,
  et retourne $f(x_1, f(x_2, \ldots, f(x_{n-1}, f(x_n, b))))$.
- `List.fold_left` :
  prend en paramètres une fonction f, un cas de base a, et une liste $[x_1 ; x_2 ; \ldots ; x_n]$,
  et retourne $f(f(f(f(a,x_1),x_2), \ldots, x_{n-1}), x_n)$.
- `List.filter` :
  prend en paramètres une fonction booléenne f et une liste $s=[x_1 ; x_2 ; \ldots ; x_n]$,
  et retourne la liste des éléments x_i de s pour lesquels f(x) est vrai.

1. Déterminer ce que retournent les commandes :

- `List.map int_of_char ['0';'2'];;`
- `List.fold_right (fun x y->x^y) ["ab";"cd";"ef"] "";;`
- `List.fold_left (-) 1000 [5;10;15;20] ;;`
- `List.filter (fun n -> n>='G') ['a';'A';'n';'4';'#';'Z'];;`

2. Définir des fonctions équivalentes à ces fonctions de référence.

Exercice 2. Utiliser la fonction `List.map` pour calculer :

- La liste des carrés des éléments d'une liste de réels.
- La liste des racines carrées des éléments d'une liste d'entiers.

Exercice 3. Utiliser `List.fold_right` puis `List.fold_left` pour calculer :

- Le nombre d'éléments d'une liste `s`.
- La somme des éléments d'une liste `s`.
- Le produit des éléments d'une liste `s`.
- L'élément minimal d'une liste `s`.

De quels types pourront être les listes passées en paramètre ?

Exercice 4. Utiliser la fonction `List.filter` pour définir une fonction qui sélectionne :

- Les éléments pairs d'une liste d'entiers.
- Les lettres majuscules d'une liste de caractères.

Exercice 5. A l'aide de fonctions d'ordre supérieur, définir les fonctions suivantes :

- `carresomme` qui calcule la somme des carrés des éléments d'une liste d'entiers.
- `n_poires` qui retourne le nombre de poires contenues dans une liste d'objets de type panier. (type défini précédemment)

Exercice 6.

1. Que fait la fonction ci-dessous ?

```
let machin s = List.fold_right (fun x y -> x::y) s [];;
```

2. Pourquoi ne peut-on pas définir la fonction ci-dessous ?

```
let truc s = List.fold_left (fun x y -> x::y) [] s;;
```

3. Utilisez la fonction `List.fold_left` pour calculer l'inverse d'une liste.

Exercice-Projet (utiliser les fonctions d'ordre supérieur autant que de besoin)

Un maraicher veut gérer ses livraisons de fruits. Il dispose de caisses dans lesquelles placer les fruits, chaque caisse pouvant être vide ou contenir un certain nombre de fruits du même type (pomme, poire, pêche, ou banane). Les caisses sont stockées empilées dans un hangar. Le maraicher charge son fourgon avec un certain nombre de caisses à livrer, en fonction des commandes de ses clients.

1. Créer les types : `caisse` et `commande` (vide ou contenant n fruits du même type), `piledecaisse` et `fourgon` (liste de caisses), `hangar` (liste de piles de caisse).

2. Créer une fonction `nombrefruits` qui donne le nombre de fruits contenus dans une caisse ou une commande.

3. Créer une fonction `aumoinscaisses` qui vérifie si une liste de caisses `p` contient au moins `n` caisses, puis créer une fonction `aumoinsfruits` qui vérifie si une liste de caisses `p` contient au moins `n` fruits.

4. Créer une fonction `ajoutcaisse` qui ajoute une caisse à une liste de caisse. Créer une fonction `ajoucaisseSi` qui ajoute une caisse à une liste seulement s'il n'y a moins de `n` caisses dans la liste (sinon la liste est inchangée).

5. Créer une fonction `repartirfruits` qui vérifie que toutes les caisses d'une pile ont au plus 30 fruits. Les caisses vides seront retirées de la pile et les caisses de plus de 30 fruits seront répartis dans deux caisses successives de la même pile.

6. Créer une fonction `organiser` qui organise un hangar en piles de 10 caisses (ou moins) d'au plus 30 fruits. Votre fonction optimise-t-elle le nombre de caisses utilisées ?

7. Le fourgon peut contenir au plus 40 caisses. Créer une fonction `remplir` qui transfère des caisses non vides du hangar pour remplir (si possible) le fourgon.

8. Créer une fonction rempliravec qui transfère uniquement des caisses d'un type de fruit `f`.

9. Créer une fonction `livrer` qui prend en paramètre une commande et remplit en conséquence le fourgon. Que fera-t-on si la commande dépasse la capacité du fourgon ?

10. Comment pourrait-on vérifier qu'il y a ce qu'il faut dans un hangar pour remplir un fourgon ? pour honorer une commande ?

11. Les bananes ne peuvent être stockées avec les autres fruits. Créer une fonction `partitionner` qui enlève les caisses de bananes d'un hangar pour les placer dans un deuxième hangar. La fonction retournera ainsi un couple de hangar (on pourra créer le type `entrepot` correspondant).
N.B. Les deux hangars devront respecter les contraintes de taille sur les piles de caisses indiquées précédemment.

12. Reprogrammer autant que de besoin les fonctions précédentes pour qu'elles fonctionnent sur un couple de hangar au lieu d'un unique hangar.