ISF - Feuille de TP 5

Exercice 1 (**Tri**)

Si pas encore fait ou corrigé, implanter la méthode de tri du quicksort (TD précédent) sur les listes.

1. Écrire une fonction `partition` qui prend en paramètre une liste et retourne deux listes : l'une contenant les éléments de `l` vérifiant le prédicat `p`, l'autre contenant les éléments de `l` ne satisfaisant pas `p`.
2. Application : tri `quicksort` dont l'idée est :
   - choisir un élément dans la liste à trier : *pivot*. On choisira comme pivot le premier élément de la liste.
   - partitionner la liste par comparaison avec le pivot (on obtient la liste des éléments strictement plus petits que le pivot et la liste des éléments $\geq$ au pivot)
   - trier les 2 sous-listes obtenues obtenues (appel récursif)
   - concaténer les sous-listes obtenues en plaçant le pivot entre les 2

   Ecrire la fonction `quicksort` qui trie une liste selon cette méthode. Vous vérifierez la terminaison de la fonction `quicksort`.
3. Donner une version de `quicksort` plus générale, paramétrée par une fonction de comparaison qui retourne 0 si ses arguments sont égaux, un entier négatif (resp. positif) si le premier argument est plus petit (resp. grand) que le 2ème.
4. Trier une liste de rationnels représentés par des couples de 2 entiers.

Exercice 2 (**Les formes géométriques**)

1. Reprendre le type `forme` du cours.
2. Définir le cercle de rayon 3,4, le rectangle de longueur 4 et de largeur 1.
3. Définir la fonction `somme_aire` de type forme `forme list -> surface` qui prend une liste de formes géométriques et calcule la somme des aires des formes. Vous en donnerez une version récursive et une version qui utilise une fonctionnelle du module `List`.

   ```
   type surface = S of float;;
   ```

4. Ecrire une fonction qui prend en argument une liste de formes et vérifie si elle contient un carré, ie un rectangle dont longueur = largeur. Vous en donnerez une version récursive et une version qui utilise une fonctionnelle du module `List`.
5. Définir le type des formes colorées en ayant au préalable défini le type couleur. Un point n'aura pas de couleur. Les couleurs possibles sont les couleurs primaires.
6. Ecrire une fonction qui extrait d'une liste de formes colorées toutes les formes d'une couleur passée en argument. Vous en donnerez une version récursive et une version qui utilise une fonctionnelle du module `List`.

Exercice 3 (**Les nombres**)

1. Définir le type des nombres qui incluera à la fois les entiers et les flottants.

2. Définir l'addition de deux nombres. Elle doit faire en sorte que la somme de deux entiers reste un entier.

3. Définir une fonction qui somme tous les éléments d'une liste de nombres. Elle retournera un nombre.

Exercice 4

Soit le type polymorphe des arbres binaires.

```
type 'a arbre_bin =
| Vide
| Noeud of 'a * 'a arbre_bin * 'a arbre_bin;;
```

1. Ecrire une fonction qui compte les valeurs stockées dans un arbre binaire. Quel est le type de la fonction ?

2. Ecrire une fonction qui calcule la hauteur d'un arbre binaire.

3. Ecrire une fonction qui vérifie que toutes les valeurs stockées dans un arbre binaire d'entiers tous sont des entiers positifs. Quel est son type ?

4. Ecrire une fonction qui prend un arbre binaire en argument et qui retourne la liste des éléments stockés dans cet arbre

   (a) en suivant un parcours infixe (GRD),

   (b) en suivant un parcours prefixe (RGD),

   (c) en suivant un parcours postfixe (GDR).

Exercice 5 (**Jouons au tarot**)

On prend la modélisation des cartes du tarot suivante

```
type couleur = Trefle | Pique | Coeur | Carreau;;
type carte_ordi =  | Roi of couleur | Dame of couleur | Cavalier of couleur
                   | Valet of couleur | Chiffre of int*couleur;;
type carte = Excuse | Atout of int | Carte_couleur of carte_ordi;;
```

On définit une main comme une liste de cartes.

1. Ecrire une fonction qui compte le nombre d'atouts dans une main.

2. Ecrire une fonction qui calcule la valeur d'une main. Rappelons qu'un roi vaut 4.5 points, une dame 3.5 points, un cavalier 2.5 points, un valet 1.5 points, chaque bout (le 1 d'atout, le 21 d'atout et l'excuse) vaut 4.5 points et toute autre carte vaut 0.5 point.

Exercice 6 (**L'école**)

Les membres de l'école peuvent être : soit des étudiants, soit des administratifs, soit des enseignants-chercheurs. Un étudiant est caractérisé par son nom et son numéro d'inscription. Un administratif est caractérisé par son nom et sa catégorie administrative, qui peut être A, B ou C. Un enseignant-chercheur est caractérisé par son nom et le nom du laboratoire de recherche auquel il est rattaché.

1. En définissant éventuellement des types intermédiaires, définissez un type `personne` pour représenter un membre de l'école.
   Dans la suite, on représentera l'école par une liste de type `personne list` appelé `école` dans la suite.

2. Écrivez une fonction `listing` qui prend une valeur de type `école` en argument et rend la liste des noms de tous les membres de l'école. Quel est son type?

3. Écrivez une fonction `separe` qui prend une valeur de type `école` en argument et rend un couple dont la première composante est la liste des noms des étudiants de l'université, la deuxième la liste des noms des administratifs et des enseignants-chercheurs. Donnez son type.