Algorithmique-Programmation : IAP1 - feuille d'exercices numéro 1

Les exercices proposés sont volontairement nombreux, de manière à ce que chacun puisse y trouver son compte. Les prendre dans l'ordre. Les exercices ou questions marqués d'une étoile sont un peu plus difficile et peuvent être passés dans un premier temps. L'étudiant y reviendra ensuite.
Pour chaque question, il est demandé d'écrire, avant toute chose, une interface (en suivant le modèle donné en cours). Vous vérifierez ce type avec la réponse de Ocaml : si il y a des différences réfléchissez un peu .....
Chaque fonction sera testée : cas nominaux, cas erronnés. Mettre les tests et leurs résultats en commentaire dans votre fichier OCaml. L'exemple de la fonction valeur absolue est donnée ci-dessous. Dans ce cas, on peut explorer les cas de tests suivants : un entier négatif (-5 par exemple), 0 et un entier positif (3 par exemple). En effet la spécification d'une telle fonction (bien connue des mathématiciens) montre bien que le résultat dépend du signe de l'entier. D'autre part, 0 est un cas de test bien légitime car est-il positif ? négatif ? une erreur dans la manipulation des signes de comparaison est si vite arrivée ....

```
(* Interface abs :
type int -> int
arg a
post abs a = valeur absolue de a
tests  -5, 0, 3*)
let abs a = if a <0 then (-a) else a;;
(* Tests
# abs (-5);;
- : int = 5
# abs 0;;
- : int = 0
# abs 3;;
- : int = 3
*)
```

Exercice 1 (**Premier contact avec OCaml**)

Pour la session OCaml suivante, indiquer les réponses de OCaml (sans machine) puis vérifier avec l'aide de OCaml.

```
# let x = 1;;
# let x = 4 in x + 2 * x;;
# x + 3;;
# let y = let z = y * y in z + 2;;
# let x = x + 1;;
# let a = 2;;
# let b = 3;;
# let a = 25 in
       let b = 7 in a + b;;
# a + b;;
# let x = 5 in  let y = 3 + x in let z = y + x in x + y + z;;
```

Exercice 2 (**Fonctions simples**)

1. Ecrire une fonction qui teste si son argument est pair. On indique que l'expression $a \ mod \ b$ retourne le reste dans la division entière de a par b.
   Remarque : **qui teste** signifie la plupart du temps **qui retourne true si la condition est vraie et false sinon**.

2. Ecrire une fonction qui retourne -1 si son argument est négatif, 0 si c'est 0 et 1 si l'argument est négatif.

3. Ecrire une fonction qui calcule le volume d'une sphère

4. Ecrire une fonction qui prend en argument 3 entiers et retourne le plus grand de ces entiers.

5. Ecrire une fonction qui ajoute de part et d'autre d'une chaîne de caractères quelconque la chaîne "!!" appelée cadre.

   Modifier la fonction de manière à ce que le cadre devienne aussi un argument de la fonction (on dit que l'on *abstrait* le cadre).

   On veut maintenant encadrer dissymétriquement. Introduire les paramètres nécessaires et écrire la nouvelle fonction.

6. Ecrire une fonction qui détermine le nombre de solutions réelles d'une équation du second degré (0 quand pas de solution réelle, 1 si racine double, 2 sinon).

Exercice 3 (**couples**)

Le rationnel $\frac{p}{q}$ sera représenté par le couple (p, q) de type `int*int`.

1. Ecrire la fonction `inverse_ratio` qui calcule l'inverse d'un nombre rationnel.

2. Ecrire la fonction qui réalise l'addition de 2 nombres rationnels. On ne cherchera pas à simplifier le rationnel résultat.

Exercice 4 (**Fonctions récursives simples**)

1. Ecrire une fonction récursive `u` telle que `u n` calcule le nième terme de la suite $(u_n)_n$ définie par : $u_0 = 1$ et $u_n = sqrt(u_{n-1} + 2), n > 0$. La fonction *sqrt* de type `float -> float` existe et calcule la racine carrée.

2. Ecrire une fonction récursive `somme` qui calcule la somme des n premiers entiers naturels : $somme\ n = \Sigma_{i=1}^{i=n} i$

3. Ecrire une fonction récursive `carre` qui calcule la somme des n premiers carrés : $carre\ n = \Sigma_{i=1}^{i=n} i^2$

4. ******* Ecrire la fonction `puissance` en utilisant une méthode dichotomique, c'est-à-dire en utilisant les résultats suivants :
   $a^{2k} = (a^2)^k$
   $a^{2k+1} = (a^2)^k\ *\ a$
   On supposera a et n entiers naturels.

5. Ecrire une fonction récursive `somme_puis` qui calcule $\Sigma_{i=0}^{i=n} x^i$, avec n un naturel quelconque et x un entier quelconque. ******* Modifier l'écriture de cette fonction de manière à optimiser les calculs.

Exercice 5 (**Listes**)

1. Ecrire une fonction qui compte le nombre d'éléments d'une liste

2. Ecrire une fonction qui compte le nombre d'éléments pairs d'une liste d'entiers

3. Ecrire la fonction `somme` qui fait la somme des nombres d'une liste de flottants.

4. Ecrire la fonction `moyenne` qui fait la moyenne arithmétique des nombres d'une liste de flottants. Ecrire une version naive de la fonction et une version ******* plus optimale (penser à calculer somme et nombre d'éléments en même temps). La fonction `float_of_int` convertit un entier en le flottant correspondant : ainsi l'expression `(float_of_int 2) +. 3.5` est une expression bien typée.

5. Ecrire la fonction `intervalle` : `intervalle (n,m)` retourne la liste des entiers compris entre `n` et `m` inclus. Si `n` est strictement supérieur à `m`, le résultat calculé est la liste vide.
   *** Généraliser cette fonction en introduisant un pas (supposé positif non nul).

6. Ecrire les fonctions `appartient` et `place` : `appartient` teste l'appartenance d'un élément à une liste et `place` donne la position d'un élément dans une liste (0 si pas dans la liste, $n > 0$ si l'élément est le $n$ième)

7. Ecrire une fonction `min` qui retourne le plus petit entier d'une liste d'entiers. Peut on l'utiliser avec uen liste de chînes de caractères ?

8. *** Ecrire une fonction qui élimine les doublons d'une liste (tout élément ne peut alors figurer qu'une seule fois dans la liste résultat)

9. *** Faire l'intersection sans doublons de 2 listes sans doublons.

10. *** Faire l'union sans doublons de 2 listes sans doublons.

Exercice 6 (**Listes triées***)
On désire maintenant manipuler des listes triées dans l'ordre croissant par exemple.

1. Ecrire une fonction qui insére un élément dans une liste triée. Le résultat obtenu doit être une liste triée dans l'ordre croissant. Vous essayerez de donner pour cette fonction une postcondition la plus formelle possible (proche d'une formule de la logique des prédicats).

2. Faire la concaténation de deux listes triées de manière à obtenir une nouvelle liste triée (on parle alors de fusion).