Programmation Logique et Fonctionnelle - TP1

CAML

1 Consignes

Lorsqu'il vous est demandé d'implanter une fonction, cela suppose une validation expérimentale de votre code, en appliquant la fonction à des jeux de données pertinents et en vérifiant l'exactitude du résultat obtenu. Si nécessaire, vous pouvez décomposer la fonction demandée en plusieurs fonctions.
Vous devez essayer de déterminer la signature de chacune des fonctions à implanter et vérifier *ensuite* qu'elle correspond à celle inférée par le compilateur. En cas de différence, vous devez identifier et comprendre votre erreur.
Les solutions de certains exercices donnés ici sont certainement disponibles quelque part sur Internet, mais cela ne vous apporterait strictement aucune compétence de saisir et d'exécuter un code que vous ne comprenez pas, ou même que vous croyez comprendre, mais sans avoir su le construire par vous-même. Si vous être vraiment bloqués, demandez plutôt des indices à votre enseignant de TP.

2 Fonctions et scalaires

2.1

Implantez une fonction `bitcounter` qui, appliquée à un entier n, retourne le nombre de bits à 1 de la représentation en base 2 de n.

2.2

Implantez une fonction `isdiv` qui, appliquée à un entier x, un entier n_1, et un entier n_2, retourne `true` si et seulement si x admet un diviseur compris entre n_1 (inclus) et n_2 (inclus).

3 Parcours de listes

3.1

Implantez une fonction `sorted` qui, appliquée à une liste d'entiers, retourne `true` si et seulement si les valeurs de cette liste sont en ordre croissant.

3.2

Implantez une fonction `ieme` qui, appliquée à une liste supposée posséder au moins $i-1$ éléments, retourne l'élément situé en position i dans cette liste.

3.3

Implantez une fonction `count` qui, appliquée à une liste l et à une valeur x, retourne le nombre d'occurrences de x dans l.

4 modification de listes

4.1

En utilisant la fonction `addfin` vue en cours, implantez une fonction `miroir` qui, appliquée à une liste l, retourne la liste miroir de l (qui contient les mêmes éléments dans l'ordre inverse).

4.2

Sans utiliser l'opérateur @, implantez une fonction `concat` qui, appliquée à deux listes, retourne le résultat de la concaténation de ces deux listes.

4.3

Implantez une fonction `adce` qui, appliquée à une liste de liste l et à une valeur x, retourne la liste obtenue en ajoutant la valeur x au début de chacune des listes contenue dans l. Par exemple, `adce [[1;2];[3;4];[]] 10` doit retourner `[[10; 1; 2]; [10; 3; 4]; [10]]`.

4.4

Implantez une fonction `findsum` qui, appliquée à une liste l d'entiers et à un entier s, retourne la liste de toutes les listes de valeurs de l ayant pour somme s. Par exemple, `findsum [1;2;3;4;5] 6;;` doit retourner `[[2; 4]; [1; 5]; [1; 2; 3]]`. (indice : en utilisant les fonctions précédentes, celle ci s'écrit en quelque lignes.)

5 fonctions d'ordre supérieur

Implantez une fonction `apply` qui, appliquée à une liste l de fonctions et à une valeur x, retourne la liste des valeurs obtenues en appliquant à x chacune des fonctions contenues dans l. Par exemple, si les fonctions `inc` et `fois2` sont définies ainsi :

```
let inc x = x+1;;
let fois2 x = x*2;;
```

alors `apply [inc;fois2;inc] 3;;` devra retourner `[4; 6; 4]`.