ENSIIE 2013/2014 Programmation fonctionnelle

Examen de rattrapage 05/02/2014 — 1h45

— **Tout appareil électronique est interdit et doit être rangé dans un sac fermé.**
— Seules les notes de cours sur papier sont autorisées.
— La rédaction doit se faire au stylo, pas au crayon.
— Lisez tout l'énoncé (2 pages) avant de commencer, lisez attentivement les questions, vérifiez que les réponses que vous proposez correspondent aux questions posées ! C a va mieux en le disant. . .
— TOUTES les fonctions doivent être précédées de commentaires indiquant les conditions d'utilisation et le résultat obtenu.

Exercice 1 [Démonstration]

On considère la structure de données suivante permettant de représenter des arbres :

```
type 'a arbre = Noeud of 'a arbre * 'a * 'arbre | Vide
```

1. Programmer une fonction `somme` qui calcule la somme de toutes les valeurs présentes dans un arbre.
2. Rappeler la définition en Ocaml de l'itérateur `map` sur les arbres. On pourra s'inspirer de l'itérateur `List.map` des listes. Utiliser cette fonction `map` pour définir une fonction `double` qui construit un arbre dont les noeuds contiennent des valeurs qui sont le double de celles de l'arbre donné en paramètre.
3. Démontrer que pour tout arbre a, `somme` (`double` a) = 2 * `somme` a.

Exercice 2 [Parenthèses]

On s'intéresse aux expressions de la forme `(((())())())`. Le but est de produire une fonction, qui étant donnée une séquence de parenthèses détermine si elle est bien formée ou non. Etre bien formée signifie que toutes les parenthèses ouvertes ont bien été refermées et que l'on n'a pas de parenthèses fermantes superflues.

1. Pour chacune des séquences suivantes, expliquer si elles sont bien formées ou non : `()()()()`, `(((())))`, `((())` et `((()))()(`.
2. On représente ces séquences de parenthèses par une liste d'éléments de type `char`. Programmer une fonction `bien_formee` qui teste si une séquence de parenthèses est bien formée.

On considère maintenant différents types de parenthèses et on veut tester la bonne formation d'une séquence de parenthèses qui contient 3 types de parenthèses `()`,`[]` et `{}`.

3. Indiquer si les séquences suivantes sont bien formées ou non : `{{{(()]}`, `({[()]})`, `{{()}}[]` et `[[))`. Expliquer pourquoi la solution précédente ne fonctionne plus (donner un contre-exemple).
4. Pprogrammer en utilisant une structure de données adéquate une fonction `bien_formee_multiple` qui teste si une liste de parenthèses est bien formée ou non.

Exercice 3 [Représentation des entiers en précision infinie]

On va représenter des grands entiers sous la forme d'une liste de nombres. A des fins de débuggage, on suppose que chaque élément de la liste ne peut contenir que des éléments strictement inférieurs à 100.

Considérons par exemple le nombre 57 230 887 (cinquante sept millons deux cents trente mille huit cents quatre-vingt sept). Ce nombre sera représenté par la liste suivante : $[87; 8; 23; 57]$. En effet, on découpe le nombre en morceaux dont la taille est inférieure à 100 et on place les 2 chiffres de poids faible dans la première position de la liste, puis les 2 chiffres suivants dans la deuxième position et ainsi de suite jusqu'à avoir représenté le nombre en entier.

1. Expliquer comment le nombre suivant 4 138 679 (quatre millions cent trente huit mille ...) peut être représenté par une telle liste. Si l'on rencontre la liste suivante $[90; 1; 23; 7]$, quel nombre représente-t-elle ? L'écrire à la fois en chiffres et en lettres.
2. Programmer une fonction d'addition `addition` de 2 grands entiers.
3. En déduire une opération `produit` de multiplication de 2 grands entiers.
4. Ecrire une fonction `valeur` qui calcule la valeur d'un grand nombre. Attention : il y a bien sûr un risque probable de débordement de capacité. Cette fonction permettra uniquement de tester nos fonctions sur de petits exemples.
5. Programmer également une fonction `decoupe` qui étant donné un entier le conditionne pour qu'il soit formaté comme un grand entier (donc comme une `int list` dont tous les élements sont inférieurs à 100).
6. Programmer une autre addition qui fonctionne quand les grands entiers sont codés des mots de poids forts vers les mots de poids faible (57 230 887 est alors représenté par [57 ;23 ;8 ;87]).