PROGRAMMATION FONCTIONNELLE
2ème session

Durée de l'épreuve : 2 heures
Documents autorisés : Cours, travaux dirigés et travaux pratiques

Attention : Toute fonction, expression, …doit être écrite en langage Caml et accompagnée de son **type.**

Exercice 1

Soit la fonction suivante

```
#let rec comb = fun
 _  []                              -> []
| a  ((b,c) :: lst)  when (b + c) = a   ->  true :: comb a lst
| a  ((b,c) :: lst)                   -> false :: comb a lst ;;
```

Questions

1. Donner le type de cette fonction
2. Donner un exemple d'appel de la fonction comb ainsi que le résultat de cet appel
3. Ecrire **une et une seul**e fonction **non récursive** équivalente à la fonction comb

Exercice 2

Une chaîne est constituée d'une suite de mots séparés par un caractère espace et terminée par un point suivant immédiatement le dernier mot

Questions

1. Ecrire une fonction ***lire_mot*** qui étant donné une chaîne renvoie le couple constitué du premier mot de la chaîne et de la chaîne privée de ce mot.

Exemple lire_mot (« le ciel est bleu. » renvoie (« le » , « ciel est bleu »)

2. Ecrire une fonction ***lire_texte*** qui, étant donné une chaîne renvoie la liste des mots la constituant. Il est conseillé d'utiliser la fonction lire_mot.

Exemple lire_texte ((« le ciel est bleu. ») donnera [« le » ; « ciel » ; « est » ; « bleu »]

Exercice 3

Les k-listes sont des listes ayant des éléments (appelés k-atomes) construits à partir

- Des valeurs Rien et Tout
- D'entiers
- De caractères
- De k-listes

La k-liste principale est de niveau 0

Une k-liste quelconque définie comme élément d'une k-liste de niveau n est de niveau (n+1). La profondeur d'une k_liste est donnée par le niveau le plus élevé de ses éléments. Nous considérons le type k-atome suivant :

```
#type kAtome   =  Rien | Tout
                | Ent of int
                | Car of char
                | Lst of kAtome list ;;
```

Questions

1. Donner un exemple de k-liste
2. Ecrire une fonction ***profondeur***, qui étant donné un k-liste renvoie la profondeur de cette k-liste
3. Ecrire une fonction à trois arguments ***substituer,*** qui étant donné deux k-atomes k1, k2 et une k-liste klst remplace toutes les occurrences de k1 dans klst par k2 et ce quelque soit son niveau dans klst.

Exercice 4

On considère des expressions arithmétiques qui peuvent être soit des entiers, soit des sommes de deux expressions arithmétiques, soit des produits de deux expressions arithmétiques. Nous nommerons **arbres d'évaluation** de telles expressions.

Questions

1. Définir le type ***arbre_eval*** pour représenter ces arbres d'évaluation.
2. Ecrire une fonction ***complexité***, qui étant donné un arbre d'évaluation renvoie le nombre d'opérations arithmétiques qu'il contient (c.à.d. le nombre d'opérations arithmétiques nécessaires pour simplifier un arbre d'évaluation en un entier).
   Avec l'arbre représentation de ((8+5)* (6 +7) on obtient 3
3. Le squelette d'un arbre définit sa géométrie : on l'obtient en supprimant toute information aux feuilles. On considère le type suivant :
   type Squelette = Vide | Jointure of Squelette * Squelette
   Ecrire une fonction ***decharne***, qui étant donné un arbre d'évaluation renvoie son squelette (de type Squelette).

Exercice 5

Nous considérons les quatre fonctions suivantes :

```
#let fois_x ->  fun x y -> x * y ;;
#let f3 = fois_x 3 ;;
#let exo_n = fun f x  -> f3 (f x) ;;
#let   exo_p = fun f  x ->  map f3 [f(x) – 2 ; f(x) – 1 ; f(x) ; f(x) + 1 ; f(x) + 2]
```

Questions

1. Donner le type de chacune de ces fonctions.
2. Donner le résultat des appels
   exo_n (function n -> n + 1) 5 ;;
   exo_n (function n -> 1) true ;;
   exo_p (function n -> n + 1) 5 ;;
   exo_p (function n -> 1) true ;;