Licence STS, semestre 3 — 2008–2009
Algorithmique et approche fonctionnelle (Info 211) — Durée : 2h
3 pages

Examen du 17 décembre 2008

Les notes de cours, de TD et de TP manuscrites ainsi que les supports de cours distribués cette année sont les seuls documents autorisés.

Veuillez lire attentivement les questions. Veuillez rédiger proprement, clairement et de manière concise et rigoureuse. Le barème est indicatif.

1 Typage (4 points)

Les fonctions suivantes sont-elles bien typées ? Si oui, donner leur type, sinon préciser pourquoi.

```
let f1 x y z = if y then x else z

let f2 x y = (x+1) :: (y-1) :: [0.]

let f3 x y z = x y z

let f4 x y z = if x+1 < y then z else print_string "ok"
```

2 Évaluation (4 points)

1. Étant donnée la fonction `mystere1: 'a list -> bool list` suivante :

   ```
   let rec mystere1 a = match a with
       [] -> []
     | [d] -> []
     | e1::e2::s -> (e1<e2)::(mystere1 (e2::s))
   ```

   donner les résultats de l'évaluation de `mystere1 [1;2;1;4;1]`.

2. Étant donnée la fonction `mystere2 : (int -> bool) -> int list -> int list` suivante :

   ```
   let rec mystere2 a b =
     match b with
         []  -> []
       | d::s ->
          if a d then d::(mystere2 a s) else (-2*d)::mystere2 a s
   ```

   donner les résultats de l'évaluation de `mystere2 (fun x->x>0) [1;2;-5;8]`.

3. Étant donné le type `'a arbre` suivant :

```
type 'a arbre = V | N of 'a * 'a arbre * 'a arbre
```

étant donnée la fonction `mystere3 : 'a arbre -> 'a -> int` suivante :

```
let rec mystere3 a b =
  match a with
      V -> 0
    | N(n,g,d) ->
        (if n>=b then 1 else 0) + (mystere3 g b) + (mystere3 d b)
```

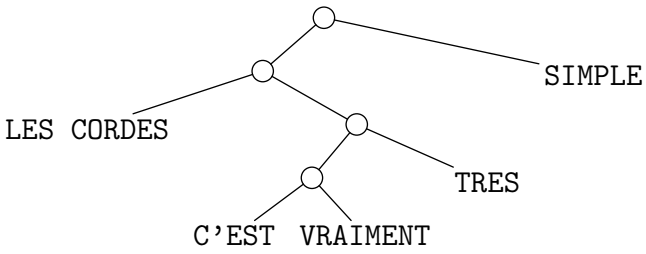
donner le résultat de l'évaluation de l'expression suivante :

```
mystere3 (N(3, N(4, N(1,V,V), V), N(3,V,V))) 2
```

3 Problème : La structure de corde (12 points)

On s'intéresse dans cette partie à l'implémentation d'une structure de donnée efficace pour représenter et manipuler des textes de grande taille (avec plusieurs millions voire milliards de caractères). Le type `string` n'est pas adapté pour deux raisons : d'une part la taille des chaînes de caractères est limitée à quelques millions de caractères, d'autre part l'opération de concaténation `^` est inefficace dès que les tailles des chaînes sont grandes.

Une alternative à ce type de donnée existe, elle est connue sous le nom de *corde*. Une corde est simplement un arbre binaire, dont les feuilles sont des bouts de texte et dont les nœuds représentent des concaténations. Par exemple, la corde suivante *représente* le texte `LES CORDES C'EST VRAIMENT TRES SIMPLE` obtenue comme la concaténation des bouts de texte `LES CORDES`, `C'EST`, `VRAIMENT`, `TRES` et `SIMPLE`.



L'intérêt des cordes est que l'opération de concaténation est immédiate puisqu'elle consiste simplement à créer un nouveau nœud dont les fils sont les deux cordes à concaténer.

Questions préliminaires (4 points). Les bouts de texte aux feuilles des cordes sont représentés par des listes de caractères, soit par le type `texte` défini par :

```
type texte = char list
```

Ainsi, le bout de texte `LES CORDES` est représenté par la liste de caractères suivante :

```
['L';'E';'S';' ';'C';'O';'R';'D';'E';'S']
```

1. Écrire la fonction `prefixe : int -> texte -> texte` telle que `prefixe i m` renvoie le texte constitué des `i` premiers caractères de `m`, en supposant $0 \leq$ `i` $\leq n$ où n est le nombre de caractères du texte `m`. Par exemple,

```
prefixe 2 ['b';'o';'n';'j';'o';'u';'r'] = ['b';'o']
prefixe 0 ['b';'o';'n';'j';'o';'u';'r'] = []
```

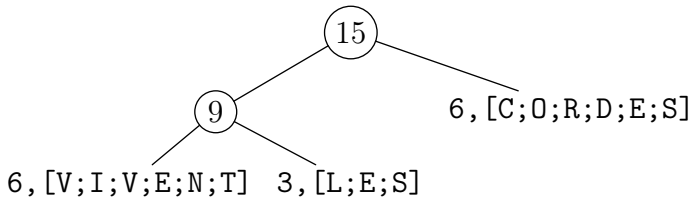
2. Écrire la fonction `suffixe : int -> texte -> texte` telle que `suffixe i m` renvoie le texte obtenu en supprimant les `i` premiers caractères de `m`, en supposant $0 \leq$ `i` $\leq n$ où n est le nombre de caractères du texte `m`.

```
suffixe 4 ['b';'o';'n';'j';'o';'u';'r'] = ['o'; 'u'; 'r']
suffixe 7 ['b';'o';'n';'j';'o';'u';'r'] = []
```

Cordes (5 points). Les cordes sont représentées par des arbres binaires dont les feuilles contiennent des valeurs de type `texte`. Afin d'accélérer le traitement de certaines opérations, on conserve à chaque feuille et à chaque nœud les longueurs des textes représentés par les cordes correspondantes. Le type des cordes est ainsi défini de la manière suivante :

```
type corde = Feuille of int * texte | Noeud of int * corde * corde
```

Ainsi, la corde suivante représente le texte `VIVENTLESCORDES` obtenue comme la concaténation des bouts de texte `VIVENT`, `LES` et `CORDES`.



1. Écrire la fonction `longueur : corde -> int` qui renvoie la longueur d'une corde.
2. En utilisant la fonction `List.length : 'a list -> int` qui renvoie la longueur d'une liste, écrire la fonction `nouvelle_corde : texte -> corde` pour construire une corde (réduite à une feuille) à partir d'un texte.
3. Écrire la fonction `concat : corde -> corde -> corde` qui construit la concaténation de deux cordes.
4. Écrire la fonction `caractere : int -> corde -> char` telle que `caractere i c` renvoie le `i`*ème* caractère de la corde `c`, en supposant que `c` est une corde non vide représentant un texte avec *au moins* `i` caractères. Vous pouvez pour cela utiliser la fonction `List.nth : 'a list -> int -> 'a` qui prend en arguments un entier `i` et une liste `l` et qui renvoie le `i`*ème* élément de `l`.

Sous-cordes (3 points).

1. Écrire la fonction `sous_corde : int -> int -> corde -> corde` telle que `sous_corde i l c` renvoie la corde correspondant au texte de longueur `l` débutant au caractère `i` de `c`. On supposera $0 \leq$ `i` $<$ `i` $+$ `l` $\leq$ `longueur c`.