info401 : Programmation fonctionnelle
TP 1 : programmes récursifs, listes

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22, poste : 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/
wiki : http://www.lama.univ-savoie.fr/wiki

Ce TP, comme les suivants, sera noté. Je ne demande aucun compte rendu à part, mais seulement un *unique* fichier Caml, commenté.

- Votre fichier doit contenir du code valide et ne doit pas provoquer d'erreur lorsqu'on l'évalue avec l'interprète Caml. (Testez avant de me l'envoyer : si votre fichier ne s'évalue pas correctement, vous perdez automatiquement 5 points sur votre note finale.)
- Votre fichier devra contenir un commentaire contenant votre nom, prénom et filière. (Idem, si ce n'est pas le cas, vous perdez 5 points sur votre note finale.)
- Pour m'envoyer votre TP, utilisez uniquement le formulaire dont l'adresse est : (lien disponible sur le wiki)
  `http://www.lama.univ-savoie.fr/~hyvernat/envoi-TP.php`

Partie 1 : Quelques compléments sur les programmes récursifs *(1h)*

Le programme

```
let rec fib n =
  if n<2
  then n
  else fib (n-1) + fib (n-2)
```

permet de calculer la suite de Fibonacci mais n'est pas très efficace. Pour essayer de comprendre ce qu'il se passe, on peut demander à Caml de *tracer* la fonction `fib` : après avoir défini la fonction, tapez "`#trace fib ;;`" dans l'interprète Caml. (N'oubliez pas le "`#`".)

Ceci permet d'afficher tous les appels intermédiaires à la fonction `fib`. Caml affiche :
- "`fib <--` n" pour dire que la fonction `fib` reçoit l'argument n,
- "`fib -->` n" pour dire que la fonction `fib` renvoie le résultat n

Pour arrêter de tracer la fonction `fib`, il suffit d'utiliser la commande "`#untrace fib ;;`".

Question 1. Tracez la fonction `fib` pour le calcul de `fib 5`. Qu'en pensez-vous ?

Remarque : il n'est pas nécessaire de mettre le code correspondant dans le fichier que vous m'envoyez : mettez seulement vos remarques en commentaire.

Question 2. Écrivez une fonction `fib_aux` de type "`int -> int*int`" qui pour l'argument n renvoie deux valeurs :
- la valeur de la fonction de Fibonacci pour n,
- la valeur de la fonction de Fibonacci pour *n-1*.

Bien entendu, vous ne devez pas utiliser la fonction `fib` pour définir `fib_aux`.

Question 3. En traçant la fonction `fib_aux`, comparer les calculs fait lors du calcul de `fib 7` et celui de `fib_aux 7`.

Comment pouvez-vous vous servir de cette fonction pour calculer la fonction de Fibonacci de manière plus efficace ?

Question 4. Essayez d'évaluer le nombre d'additions utilisées pour calculer Fibonacci avec `fib` et avec `fib_aux`. Qu'en pensez-vous ?

Question 5. (Bonus) Écrivez une fonction `fib3` de type "`int -> int -> int -> int`" qui fait la chose suivante :

- son premier argument est l'entier n pour lequel on veut calculer la fonction de Fibonacci.
- ses deux autres arguments sont des *accumulateurs* qui permettent de conserver des valeurs successives (pour *i-1* et i) de la fonction de Fibonacci. Lorsque n diminue, ces deux arguments augmenteront...

  Remarque : cela veut dire que pour calculer la fonction de Fibonacci sur l'entier n, il faudra appeler "`fib3 0 1` n".

Tracez cette fonction lors du calcul de "`fib3 15`". Que constatez-vous ?

À votre avis, quelle est la meilleur méthode pour calculer la fonction de Fibonacci : en utilisant `fib2` ou en utilisant `fib3` ?

Partie 2 : Manipulation des listes *(1h)*

Question 1. (facultatif) Reprenez quelques fonctions du TD sur les listes et programmez les en utilisant

- une définition directe,
- la fonction `List.fold_right`.

Question 2. Programmez la fonction `renverse` comme nous l'avons fait en TD, puis comparer l'efficacité de votre fonction avec celle de la fonction `List.rev`, qui fait la même chose.

Expliquez la démarche que vous suivez pour faire cette comparaison en mettant des commentaires dans votre fichier Caml.

Question 3. La fonction `List.map` permet d'appliquer une fonction à tous les éléments d'une liste.

Écrivez une fonction `pam` qui prend en arguments :

- une valeur,
- une liste de fonctions

et qui renvoie la liste des valeurs obtenues en appliquant les fonctions de la liste à la valeur. Par exemple :

```
pam 3 [ (fun x -> x+1) ; (fun x -> x*x) ; (fun x -> 42-x) ]
```

donnera `[4 ; 9 ; 39]`.

Essayez de deviner le type de cette fonction sans regarder le résultat donné par Caml.

Question 4. Écrivez la fonction `pam` uniquement à partir de la fonction `map`.

Partie 3 : Deux algorithmes de tri pour les listes *(2h)*

Le *tri par insertion* et le *tri fusion* sont deux algorithmes de tri naturellement récursifs.

Le tri par insertion fonctionne de la manière suivante :

- la liste vide est triée,
- pour trier la liste "`a::l`", on commence par trier (récursivement) la liste `l`, puis on *insère* l'élément `a` dans le résultat.

Le tri fusion fonctionne de la manière suivante :

- la liste vide est triée,
- pour trier une liste non-vide, on sépare la liste en deux parties de longueur égale (à un élément prêt), on trie (récursivement) ces deux listes, puis on *fusionne* les deux résultats.

Question 1. Programmez une fonction `insertion` qui permet d'insérer un élément à sa place dans une liste triée.

Question 2. Programmez ensuite le tri par insertion : `tri_insertion`.

Question 3. Programmez une fonction `fusionne` qui permet de fusionner deux listes triées en une seule liste triée.

Question 4. Programmez une fonction `separe` qui permet de séparer une liste quelconque en deux listes de taille équivalente.

Question 5. Programmez maintenant le tri fusion : `tri_fusion`.

Question 6. Comparez l'efficacité de vos deux algorithmes. Qu'en pensez-vous ?

Expliquez la démarche que vous suivez pour faire cette comparaison en mettant des commentaires dans votre fichier Caml.

Question 7. Imaginez la situation suivante : vous avez une liste de points dans le plan, c'est à dire une liste de paires de flottants (type "`float*float list`").

Vous aimeriez trier cette liste selon la hauteur de points, c'est à dire suivant leur seconde coordonnée.

Comment pourrait-on profiter des capacités de Caml à manipuler des fonctions pour réécrire les algorithmes de tri de manière à les faire fonctionner dans ce cadre là.

Qu'en pensez-vous ?

Question 8. *(Bonus)* Un algorithme de tri est dit *stable* quand il ne modifie pas l'ordre relatifs des éléments "égaux". (Par exemple, lorsque vous triez vos fichiers par taille croissante, vous aimeriez que pour les fichiers de même taille, l'ordre soit toujours l'ordre alphabétique.)

Est-ce que vos programmes de tri par insertion et de tri fusion sont stables ?

Sinon, pouvez-vous les rendre stables ?

Partie 4 : Bonus, si vous vous ennuyez...

Programmez une fonction `premier : int -> int list` qui calcule tous les nombres premiers entre 0 et un nombre entier.

```
# premier 10 ;;
- : int list = [2; 3; 5; 7]
```