Travaux Pratiques ML No.2

Cours d'Informatique de Deuxième Année

—Deug SSM M4—

Aujourd'hui...

Le but de ce *TP* est de manipuler une structure très utile en *Caml-Light* : la liste. On introduit d'abord la notion de *Pattern Matching*, c'est-à-dire la *correspondance de motif*.

Pattern Matching et introduction aux listes

▸ **Exercice 2**. Ecrire une fonction qui teste si un entier est nul.

. . . . . .

▸ **Exercice 3**. Ecrire une fonction qui teste si un couple a ses deux composantes égales.

. . . . . .

▸ **Exercice 4**. Ecrire la fonction `fst` qui renvoie le premier élément d'un couple.

. . . . . .

Une liste en *Caml-Light* est un ensemble d'éléments de même type qui vérifie la syntaxe suivante : `x::l` est la liste qui commence par l'élément `x` et dont le reste de la liste est `l`; `[1;2;3;4]` est la liste d'entiers composée des éléments `1`, `2`, `3` et `4`.

Il existe un certain nombre de fonctions permettant de manipuler les listes, parmi lesquelles "`::`" qui permet d'ajouter un élément en début de liste et "`@`" qui permet la concaténation de deux listes.

▸ **Exercice 5**. Reprogrammez la fonction `@` en notation infixe.

. . . . . .

▸ **Exercice 6**. Donnez la fonction `list_length` qui calcule la longueur d'une liste

. . . . . .

▸ **Exercice 7**. Donnez la fonction `rev` qui retourne une liste.

. . . . . .

▸ **Exercice 8**. Donnez la fonction qui renvoie le premier élément d'une liste.

. . . . . .

▸ **Exercice 9**. Donnez une fonction qui renvoie le dernier élément d'une liste.

. . . . . .

▸ **Exercice 10**. Réécrire la fonction `map`, qui étant donnés une liste `l` et une fonction `f`, renvoie la liste des images des éléments de `l` par `f`.

. . . . . .

▸ **Exercice 11**. Etant donné un prédicat `P` et une liste `l`, donnez une fonction `pred_list` qui renvoie la liste des `P x`, où `x` parcourt tous les éléments de `l`.

. . . . . .

▸ **Exercice 12**. Utilisez `pred_list` pour tester quels éléments d'une liste sont pairs.

. . . . . .

▸ **Exercice 13**. Donnez une fonction `sum` qui renvoie la somme des éléments d'une liste.

. . . . . .

▸ **Exercice 14**. Ecrire une fonction qui fusionne deux listes triées. En déduire la fonction de tri par fusion sachant que la fonction de partage d'une liste peut s'écrire :

```
#let partage l = partl ([],[]) l
#        where rec partl = fun
#          co [] -> co
#        | (a,b) (c::m) -> partr (a@[c],b) m
#        and
#        partr = fun
#          co [] -> co
#        | (a,b) (c::m) -> partl (a,b@[c]) m;;
```

. . . . . .

. . . . . .

. . . . . .

▸ **Exercice 15**. Ecrire une fonction `apply` qui prend en argument une liste de fonctions `lf` et une liste d'arguments `la` et qui applique le premier élément de `lf` au premier élément de `la`, ... ; la fonction devra retourner la liste des résultats.

. . . . . .

. . . . . .

Exercices supplémentaires

▸ **Exercice 16**. Refaire l'exercice sur le tri fusion en écrivant une fonction `ptri_fusion` triant une `'a list`. On veillera à ce que la fonction prenne en argument le prédicat de comparaison de deux éléments du type `'a`.

▸ **Exercice 17**. Refaire la fonction `partage` du tri fusion de telle sorte que `partage [1;2;3;4;5]` renvoie le couple `([1;2;3],[4;5])`.