Travaux Pratiques sur Caml Light n^{o}2

Cours d'Informatique de l'École Centrale d'Électronique

—Ingénieur Deuxième Année—

Ce TP traite des différents points suivants :

- les listes ;
- le *pattern matching* ;
- la notion de fonctionnelle d'itération.

▸ **Exercice 1.** *Donnez une fonction qui teste si un entier est nul.*

▸ **Exercice 2.** *Donnez une fonction qui teste si un couple a ses deux composantes égales.*

▸ **Exercice 3.** *Donnez une fonction* `fst` *qui renvoie le premier élément d'un couple.*

Une liste 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`. Par exemple, `[1;2;3;4]` est la liste d'entiers composée des éléments `1`, `2`, `3` et `4`. Il existe un certain nombre d'opérateurs permettant de manipuler les listes, parmi lesquels "`::`" qui permet d'ajouter un élément en début de liste et "`@`" qui permet la concaténation de deux listes.

▸ **Exercice 4.** *Reprogrammez la fonction* `@` *en notation préfixe.*

▸ **Exercice 5.** *Donnez la fonction* `list_length` *qui calcule la longueur d'une liste.*

▸ **Exercice 6.** *Donnez la fonction* `list_rev` *qui renverse l'ordre des éléments d'une liste.*

▸ **Exercice 7.** *Donnez une fonction* `list_sum` *qui renvoie la somme des éléments d'une liste.*

▸ **Exercice 8.** *Donnez une fonction* `list_merge` *qui fusionne deux listes triées en une unique liste triée. En déduire la fonction* `merge_sort` *de tri par fusion de deux listes sachant que la fonction* `list_divide` *de partage en deux d'une liste peut s'écrire :*

```
#let list_divide l = left_part ([],[]) l where
   rec left_part = fun
            co [] -> co
   | (a,b) (c::m) -> right_part (a@[c],b) m
```

```
 and
       right_part = fun
            co [] -> co
   | (a,b) (c::m) -> left_part (a,b@[c]) m;;
list_divide : 'a list -> 'a list * 'a list = <fun>
```

Par exemple, on a :

```
#list_divide [5;7;3;8;2;1];;
- : int list * int list = [5; 3; 2], [7; 8; 1]
```

▸ **Exercice 9.** *Réécrivez la fonctionnelle* `map`*, qui étant données une liste* `l` *et une fonction* `f`*, renvoie la liste des images des éléments de* `l` *par* `f`*.*

Les exercices suivants sont à faire à l'aide des fonctionnelles sur les listes `list_it` et `map` :

```
#let list_it a f = list_it_rec where rec list_it_rec = function
    [] -> a
 | x::l -> f x (list_it_rec l);;
list_it : 'a -> ('b -> 'a -> 'a) -> 'b list -> 'a = <fun>
#map;;
- : ('a -> 'b) -> 'a list -> 'b list = <fun>
```

▸ **Exercice 10.** *Donnez une fonction qui à la liste* $[x_1; \ldots; x_n]$ *associe* $\sum_{i=1}^{n} x_i$.

▸ **Exercice 11.** *Donnez une fonction qui renvoie la conjonction des éléments d'une liste.*

▸ **Exercice 12.** *Donnez une fonction* `list_rev` *qui renverse l'ordre des éléments d'une liste (en utilisant la fonctionnelle* `list_it`*).*

▸ **Exercice 13.** *Donnez une fonction* `list_flat` *qui à une liste de listes associe la liste des éléments des listes. Par exemple :*

```
#list_flat [[3;5];[5;7;1];[9;4]];;
- : int list = [3; 5; 5; 7; 1; 9; 4]
```

▸ **Exercice 14.** *Donnez une définition de la fonctionnelle* `map` *en utilisant la fonctionnelle* `list_it`.

▸ **Exercice 15.** *Donnez une fonction* `list_subsets` *prenant en argument une liste* `l` *et retournant la liste des parties de* `l`*. On utilisera avantageusement les fonctionnelles* `list_it` *et* `map`.