TP1 : Programmation récursive sur les listes

Par convention, les identificateurs des variables mathématiques sont en police mathématique n, les identificateurs des variables informatiques sont en police bureautique `n`, étant entendu que le même identificateur, peu importe sa police, fait référence à la même variable, la police change en fonction de son contexte d'utilisation.

Programmation de certaines primitives sur les listes

L'idée de ce paragraphe est de mettre en place les instructions classiques sur les listes à partir des commandes de bases que sont :

- la liste vide `[]`,
- l'ajout-en-tête d'une liste `::`,
- la tête d'une liste `List.hd`,
- la queue d'une liste `List.tl`.

Par exemple, mettons en place une fonction qui calcule la longueur d'une liste :

```
let rec longueur l = match l with
    | [ ] -> 0
    | t::q -> 1 + longueur q
;;
```

Dans la suite, les commandes sur les listes se limitent aux commandes de bases.

1. Écrire en OCaml, une fonction qui détermine l'appartenance d'un élément à une liste :
   `let rec appartient x l`
2. Écrire en OCaml, une fonction qui détermine le nombre d'apparitions d'un élément dans une liste :
   `let rec occurence x l`
3. Écrire en OCaml, une fonction qui extrait le dernier élément d'une liste :
   `let rec dernier l`
4. Écrire en OCaml, une fonction qui ajoute un élément en queue de liste :
   `let rec ajout_fin x l`
5. Écrire en OCaml, une fonction qui extrait le i^{e} élément d'une liste :
   `let rec i_eme i l`
6. Écrire en OCaml, une fonction qui supprime le i^{e} élément d'une liste :
   `let rec supprime i l`
7. Écrire en OCaml, une fonction qui supprime toutes les occurrences d'un élément dans une liste :
   `let rec supprime_tout x l`
8. Écrire en OCaml, une fonction qui insère un élément dans une liste en i^{e} position :
   `let rec insere x i l`
9. Écrire en OCaml, une fonction qui concatène deux listes :
   `let rec concatene l1 l2`

10. Écrire en OCaml, une fonction qui effectue la concaténation inverse de deux listes :
    ```
    let rec concatene_inverse l1 l2
    ```
    Par exemple, `concatene_inverse [2;4;7] [1;3]` retourne `[7;4;2;1;3]` .
    En déduire une fonction en OCaml qui calcule le miroir d'une liste :
    ```
    let miroir l
    ```
    Par exemple, `miroir [7;4;2;1;3]` retourne `[3;1;2;4;7]` .

Opérations « algébriques » sur les listes

11. Écrire en OCaml une fonction d'addition de deux listes :
    ```
    let rec addition l1 l2
    ```
    Par exemple, `addition [1;3;2;6;7] [2;2;4]` retourne
    `[3(=1+2);5(=3+2);6(=2+4);6;7]` .
12. Écrire en OCaml une fonction de fusion de deux listes triées par ordre croissant
    ```
    let rec fusion l1 l2
    ```
    Par exemple, `fusion [1;3;4;7;8] [2;4;5;9]` retourne
    `[1;2;3;4;4;5;7;8;9]` .

Algorithmes classiques sur les listes

13. Écrire en OCaml, une fonction qui détermine le maximum des éléments d'une liste, les éléments de la liste appartenant à un ensemble ordonné :
    ```
    let rec maximum l
    ```
14. Réfléchir à une fonction récursive qui détermine le second maximum, s'il existe, des éléments d'une liste, les éléments de la liste appartenant à un ensemble ordonné.
    ```
    let rec second_maximum l
    ```
15. Écrire en OCaml, une fonction qui calcule la somme des éléments d'une liste, les éléments de la liste appartenant à un ensemble muni d'une addition :
    ```
    let rec somme l
    ```

Algorithme de Hörner

L'algorithme de Hörner permet de calculer la valeur d'un polynôme $P = \sum_{k=0}^{n} a_k X^k \in \mathbb{R}[X]$ en une valeur $x \in \mathbb{R}$ à l'aide du calcul suivant :

$$P(x) = a_0 + x(a_1 + x(\ldots + x(a_{n-2} + x(a_{n-1} + xa_n))\ldots)).$$

On représentera le polynôme P sur OCaml par la liste de ses coefficients $[a_0; a_1; \ldots; a_{n-1}]$.

16. Écrire en OCaml une fonction d'évaluation d'un polynôme P en une valeur x utilisant l'algorithme d'Hörner :
    ```
    let rec evaluation P x
    ```

Coefficients binomiaux

17. Écrire en OCaml une fonction de calcul du coefficient binomial $\binom{n}{p}$, $(n,p) \in \mathbb{N}^2$:

```
let rec binomial n p
```

18. Écrire en OCaml une fonction qui, étant une ligne du triangle de Pascal représentée sous forme d'une liste, calcule la ligne suivante :

```
let rec ligne_suivante l
```

Par exemple, `ligne_suivante [1;3;3;1]` retrourne `[1;4;6;4;1]` .

19. Écrire en OCaml une fonction qui pour un entier naturel n , renvoie la n^{e} ligne du triangle de Pascal.

```
let rec pascal n
```

Par exemple, `pascal 4` retourne `[1;4;6;4;1]` .