

Feuille 5 : Enregistrements, types récurifs et récursivité terminale

Exercice 5.1 Réécrire les opérations sur les complexes le type suivant.

```
type complex = { real : float; imag : float; }
```

Exercice 5.2 1. Écrire un type `date` de type enregistrement, contenant trois champs entiers : `day`, `month`, `year`.

2. Définir une variable `today` contenant la date d'aujourd'hui.

3. Écrire un prédicat `date_infeg` qui s'applique à deux dates `date1` et `date2` et qui retourne `true` si `date1` précède `date2`.

Exercice 5.3 1. Définir un type `'a mylist` permettant de représenter les listes d'éléments de type `'a`.

2. Écrire la fonction `mylist_length` qui prend en argument une liste de type `mylist` et qui retourne le nombre d'éléments de la liste. Exemple :

```
# len Nil;;
- : int = 0
# len (C(0, C(1, C(3, C(4, C(5, Nil))))));;
- : int = 5
```

3. Écrire la fonction `interval_list n p` de type `int -> int -> int mylist` qui retourne la liste ordonnée des entiers contenus dans l'intervalle $[n, p]$. Exemple :

```
# interval_list 4 1;;
- : int mylist = Nil
# interval_list 2 5;;
- : int mylist = C(2, C(3, C(4, C(5, Nil))))
```

4. Écrire la fonction `map`, qui prend en argument

- une fonction de type `'a -> 'b`, et
- une liste de type `'a mylist`

et telle que `map f l` renvoie la liste de type `'b mylist` obtenue en appliquant `f` à chaque élément de `l`. Exemple :

```
# map (fun x -> x+10) (C(0, C(1, C(3, C(4, C(5, Nil))))));;
- : int mylist = C(10, C(11, C(13, C(14, C(15, Nil))))
```

5. Écrire une fonction `filter pred l` de type `('a -> bool) -> 'a mylist -> 'a mylist` qui retourne la listes des éléments de `l` qui vérifient le prédicat `pred`. Exemple :

```
# filter (fun x -> x mod 2 = 0) (C(0, C(1, C(3, C(4, C(5, Nil))))));;
- : int mylist = C(0, C(4, Nil))
```

Exercice 5.4 1. Les entiers de Peano sont une représentation des entiers naturels. Ils sont construits à partir de 0 en appliquant la fonction successeur. Par exemple, 1 est le successeur de 0, et 2 est le successeur

du successeur de 0.

Pour un élément $m \in \mathbb{N}$ on peut définir les suites $(m + n)_{n \in \mathbb{N}}$ et $(m \times n)_{n \in \mathbb{N}}$ comme il suit :

$$(m + n)_{n \in \mathbb{N}} = \left\{ \begin{array}{l} m + 0 = m \\ \forall n \in \mathbb{N} \quad m + S(n) = S(m + n) \end{array} \right\} \quad (1)$$

$$(m \times n)_{n \in \mathbb{N}} = \left\{ \begin{array}{l} m \times 0 = 0 \\ \forall n \in \mathbb{N} \quad m \times S(n) = (m \times n) + m \end{array} \right\} \quad (2)$$

- a. Proposer un type OCaml appelé `peano` pour représenter les entiers de cette façon. Le type aura deux constructeurs : un pour représenter 0, l'autre pour représenter le successeur d'un entier.
- b. Écrire la fonction d'addition sur ces entiers.
- c. Écrire la fonction de multiplication sur ces entiers.
- d. Écrire les fonctions de conversion `peano_of_int` et `int_of_peano`.

Exercice 5.5 Écrire une version récursive terminale de la fonction factorielle.

Exercice 5.6 Écrire une version récursive terminale de la fonction `sum` de type `int -> int -> int` qui retourne la somme de entiers de l'intervalle `[n, p]`.

Exercice 5.7 Écrire une version récursive terminale de la fonction `mylist_length` vue au chapitre sur les types récursifs.

Exercice 5.8 Écrire une version récursive terminale de la fonction `interval_list` vue au chapitre sur les types récursifs.