1 Somme de types

On définit le type `nombre` qui contient les entiers et les flottants (cf. cours).

Écrire le type nombre avec ses deux étiquettes : `Entier` et `Reel`.

Écrire les fonctions `float_of_nombre` et `nombre_of_float` qui permettent de passer du type `float` au type `nombre`.

Écrire la fonction `plus` qui prend deux nombres en arguments et retourne leur somme sous la forme d'un nombre.

Utiliser le type nombre, les fonctions `nombre_of_float` et `plus` pour calculer la somme de deux décimaux a et b supposés définis auparavant (on pourra prendre $a = 1.$ et $b = 2.$).

Ecrire la fonction qui prend une liste de nombres en argument et retourne la somme des éléments, sous la forme d'un nombre.

2 Ecrire des termes

Dans cet exercice, nous allons représenter des formules algébriques à l'aide de termes. On veut pouvoir modéliser des formules du type $2 + x + 4 * (y - 2)$

Attention, l'objectif n'est pas de calculer la valeur de la formule mais seulement de la modéliser !

Proposer un type formule (vu en cours) permettant de modéliser une formule.

Trouver les étiquettes représentant :

1. les valeurs numériques.
2. les inconnues.
3. les opérations

A l'aide de ce type transformer en Caml les expressions mathématiques suivantes :

1. $5 * x * y - 3 * x + y - 4$
2. $2 + 5 * x - 3 * y$
3. $(2 + 5) * (x - 3) * y$
4. $x * x - 3 * y + 2$

Pensez à bien construire les arbres de représentation de la formule.

Retrouver la formule mathémathique correspondant aux expressions Caml suivantes

1.
```
Plus(Moins(Fois(Coef 8, Fois(Inc "x", Inc "y")), Coef 5),
Moins(Fois(Coef 6, Fois(Inc "x", Inc "x")),
Fois(Coef 4, Fois(Inc "x", Inc "z"))));;
```

2.
```
Plus(Fois(Moins(Inc "y", Fois(Coeff 4, Inc "y")), Inc "x"),
Plus(Fois(Coef 6, Fois(Inc "y", Inc "x")),
Moins(Coef 3, Fois(Inc "x", Coef 2))));;
```

3 Ensembles ordonnés

Dans cet exercice, nous vous proposons de construire une structure permettant de stocker des ensembles ordonnés d'entiers. Cette structure pourra s'apparenter au type liste que vous avez déjà vu.

Note : les entiers ne sont pas forcément rangés dans l'ordre croissant, {5,4,8,9} est un ensemble ordonné d'entiers. {4,5,9,8} en est un autre.

Un ensemble ordonné est défini de manière récursive comme étant
– soit un couple (entier premier, ensemble restant) : premier désignant le premier élément de l'ensemble et restant l'ensemble privé du premier élément.
– soit un ensemble vide.

Un type ensemble ordonné est donc défini par `type ens_ordo= Chaine of entier * ens_ordo | Vide;;`

1. Représenter en CAML les ensembles ordonnés suivant {}, {5}, {3,2,1} en utilisant la structure proposée
2. Écrire une fonction nombre_elements qui compte le nombre d'éléments contenus dans un ensemble passé en paramètre
3. Écrire une fonction ajout `ens_ordo -> int -> ens_ordo` qui permet d'ajouter à l'ensemble ordonné passé en premier paramètre l'élément passé en second paramètre.
4. Écrire une fonction premier `ens_ordo -> int` qui renvoie le premier élément de l'ensemble ordonné.

4 Arbres généalogiques

Dans cet exercice, nous allons représenter des arbres généalogiques.

Un arbre généalogique est défini récursivement par le nom d'une personne ainsi que la liste des arbres généalogiques de ses enfants.

On définit le type arbre_gen par `type arbre_gen = Personne of (string * arbre_gen list);;`

Représenter la famille Dupont sachant que Pierre, le grand-pere à trois enfants Paul, Victor et Julie. Paul a trois enfants Elisabeth, Alice et Bertrand. Victor a un fils unique Eric et Julie n'a pas d'enfants.

Écrire une fonction `nb_fils : arbre_gen -> int` qui retourne le nombre de fils de la personne en haut de l'arbre genealogique. (Par exemple `nb_fils dupont = 3`)

Écrire une fonction `nb_petit_fils : arbre_gen -> int` qui fonctionne de manière équivalente (`nb_petit_fils Dupont=4`)

Ecrire une fonction descendance `arbre_gen -> int -> int` qui fournit le nombre de descendants de la $k^{\text{ième}}$ génération, k étant le second paramètre de la fonction descendance.

– descendance (dupont,0) donne 1
– descendance (dupont,1) donne 3
– descendance (dupont,2) donne 4
– descendance (dupont,3) donne 0
– descendance (dupont,n) donne 0 pour $n > 3$