IAP1 - Feuille de TD types concrets

Exercice 1 (**Les nombres (si pas déjà fait)**)

1. Définir le type des nombres qui incluera à la fois les entiers et les flottants.
2. Définir l'addition de deux nombres.
3. Définir une fonction qui somme tous les éléments d'une liste de nombres. Elle retournera un nombre.
4. Définir une relation d'ordre sur les nombres.

Exercice 2 (**Les entiers naturels**)

On définit le type des entiers naturels de la façon suivante

```
type nat = Z | S of nat;;
```

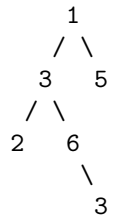
1. Ecrire l'entier 5 de type `nat`.
2. Ecrire la fonction `plus` qui fait l'addition de 2 naturels.
3. Ecrire la fonction `mult` qui fait la soustraction de 2 naturels.
4. Ecrire le prédicat `inf` qui teste si un entier naturel est inférieur ou égal à un autre entier naturel.
5. Ecrire une fonction qui calcule le plus grand de deux entiers naturels.

Exercice 3 (**Des arbres**)

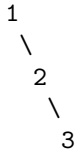
On définit le type des arbres d'entiers :

```
type arbreb = Vide | Racine of  arbreb * int * arbreb;;
```

1. Définir des exemples d'arbres avec 1 entier, 2 entiers, 3 entiers
2. Définir l'arbre suivant. Il prendra le nom `exemple`. L'arbre vide n'est pas représenté.



3. Définir l'arbre suivant. Cette forme d'arbre est appelée *peigne*.



4. Ecrire la fonction qui calcule la liste des valeurs entières rangées dans un arbre binaire. Quel est son type ?

5. Ecrire une fonction qui teste si un arbre est un peigne. Un arbre vide sera considéré comme un peigne.

6. Écrire une fonction qui remplace dans l'arbre toutes les occurrences de la valeur r par la valeur v.

7. Reprendre les questions de cet exercice de manière à définir le type des arbres binaires paramétré par le type des valeurs.

8. Ecrire une fonction qui retourne le sous-arbre droit d'un arbre.

9. Ecrire une fonction qui retourne le sous-arbre droit du sous-arbre gauche d'un arbre.

10. Généralisons ! (extrait du contrôle 2005) À un arbre on peut associer la liste des choix faits pour désigner un sous-arbre. On notera par `D` le choix *Aller à droite* et par `G` le choix *Aller à gauche*. Par exemple, dans l'arbre `exemple`, la liste des choix `[G;D]` désigne le sous-arbre `Racine (Vide, 6, Racine(Vide, 3, Vide))`. La liste de choix vide désigne l'arbre lui-même.

    Définir la fonction `sous_arbre` qui prend en paramètre une liste de choix `choix` (de type `direction list`) et un arbre et retourne le sous-arbre désigné par la liste des choix `choix`. Si l'arbre désigné n'existe pas, la fonction échouera. Donner son type.

    On définit le type `direction` par : `type direction = G | D;;`