# TP 4 : Définitions inductives (et récursives)

## 1 Définitions inductives

On rappelle la définition inductive des entiers naturels dans Coq :

```
Inductive nat : Set := O : nat | S : nat -> nat.
```

**1-** Définir une fonction `myplus` réalisant l'opération d'addition sur les entiers naturels. Cette fonction devra calculer par récurrence sur son ***deuxième*** argument. Tester le comportement de cette fonction sur quelques exemples simples avec la commande `Eval compute in`.
**2-** Démontrer l'associativité de `myplus` :

```
forall a b c : nat, myplus (myplus a b) c = myplus a (myplus b c).
```

**3-** Démontrer la commutativité de `myplus` :

```
forall a b : nat, myplus a b = myplus b a.
```

En cas de difficulté, on pourra essayer de démontrer les lemmes intermédiaires suivants :

```
myplus_Sn_m : forall n m :nat, myplus (S n) m = S (myplus n m).
myplus_O_m : forall m:nat, myplus O m = m.
```

**4-** Définir une fonction de sommation $\Sigma$ de 0 à n prenant en argument une fonction $f$ de type `nat` $\rightarrow$ `nat` et une borne $n$ : `nat` et calculant $\Sigma_{i=0}^{n} f(i)$

```
sommation : (nat -> nat) -> nat -> nat
```

**5-** Démontrer que

$$2 * \Sigma_{i=0}^{n}\ i = n * (n + 1).$$

On pourra utiliser la tactique `ring` (faire `Require Export ArithRing.` pour charger cette tactique) ainsi que les théorèmes déjà prouvés dans la bibliothèque standard de Coq en particulier les deux lemmes suivants :

```
plus_n_Sm : forall n m : nat, S (n + m) = n + S m
plus_n_O : forall n : nat, n = n + 0
```

## 2 Nombre de pièces

On souhaite maintenant démontrer qu'avec un nombre infini de pièces de 3 et 5 euros on peut faire l'appoint pour tout montant supérieur à 8 euros. Il s'agit donc de démontrer un théorème de la forme suivante :

$$\forall m : \texttt{nat}, \exists i : \texttt{nat}, \exists j : \texttt{nat}, 8 + m = 5 * i + 3 * j.$$

**6-** Précisez le sens des variables $m$, $i$ et $j$.
**7-** Proposez un mécanisme de démonstration sur papier pour ce théorème.
**8-** En faire une démonstration en Coq.

# 3 Listes

On rappelle la définition des listes polymorphes :

```
Inductive liste (A:Set) : Set :=
| nil  : liste A
| cons : A -> liste A -> liste A.
```

**9-** Construire la fonction `map` qui applique une fonction $f$ de $A$ vers $B$ à chaque élément de la liste.
**10-** Définir une fonction de composition o permettant de composer les fonctions $f$ et $g$. Si $f : A \rightarrow B$ et $g : B \rightarrow C$, g o f sera de type $A \rightarrow C$.
**11-** Démontrer le théorème suivant :

$$\forall A : \texttt{Set}, \forall f : A \rightarrow A, \forall g : A \rightarrow A, \forall l : (\texttt{liste}\ A), \texttt{map}\ (f\ o\ g)\ l = \texttt{map}\ f\ (\texttt{map}\ g\ l)$$

**12-** S'il reste du temps, écrire une fonction de concaténation de listes `append` et une fonction de retournement de liste `reverse` et démontrer que

$$\forall A : \texttt{Set}, \forall l\ m : (\texttt{liste}\ A), \texttt{reverse}\ (\texttt{append}\ l\ m) = \texttt{append}\ (\texttt{reverse}\ m)\ (\texttt{reverse}\ l)$$