TP 2

Ce TP est à déposer à la fin de cette séance sur Moodle. Vous pouvez également déposer une version améliorée de votre TP jusqu'à la fin de cette semaine.

Fichier à récupérer depuis moodle : `tp2.mlw`

Exercice 1 : terminaison

Dans le module `Ex1`, **spécifier** et compléter les deux sous-programmes suivants afin de prouver avec Why3 qu'ils terminent.

```
let f1 () =
    let n = ref 0 in
    while !n < 100  do
      n := !n + 1
    done;
    !n

  let f2 () =
    let n = ref 100 in
    while !n > 0  do
      n := !n - 1
    done;
    !n
```

Exercice 2 : tableaux

1. Dans le module `Ex2`, spécifier en Why3 un sous-programme `somme_tab` calculant la somme des éléments d'un tableau d'entiers positifs. La spécification de `somme_tab` utilisera la fonction `sum` du module `ArraySum` de la bibliothèque des tableaux de Why3.

   ```
   (** [sum a l h] is the sum of [a[i]] for [l <= i < h] *)
   function sum (a: array int) (l h: int) : int = …
   ```

2. Tester cette spécification en complétant le module de test `Ex2Test`.
3. Programmer le sous-programme `somme_tab`.
4. Avec Why3, prouver la correction de `somme_tab`. Avant de demander à Why3 de prouver votre sous-programme, vous utiliserez la stratégie « Split VC» pour comprendre les différentes conditions de vérification de ce sous-programme.

Exercice 3 : division entière

L'objectif de cet exercice est de spécifier, programmer et prouver un sous-programme effectuant la division entière de deux entiers strictement positifs.

1. Compléter le module `Ex3` en spécifiant le sous-programme `division`.
2. Tester cette spécification à l'aide du module de test.
3. Programmer `division` en utilisant la méthode d'Euclide pour calculer la division entière de l'entier `a` par `b`. Pour rappel, il s'agit de soustraire `b` à `a` tant que cela est possible.
4. Avec Why3, prouver la correction et la terminaison de `division`.

Exercice 4 : PGCD

1. Compléter le module `Ex4` en spécifiant le sous-programme `pgcd` qui calcule le PGCD de deux entiers naturels. La spécification utilisera l'extrait suivant de la bibliothèque `number.Gcd`.
   ```
   function gcd (x y : int) : int
   ```
2. Programmer `pgcd` en écrivant une version itérative de l'algorithme d'Euclide (rappelée ici : https://fr.wikipedia.org/wiki/Algorithme_d%27Euclide) calculant le PGCD de deux entiers naturels. Les extraits utiles de ce page sont rappelés ci-dessous.
   « Le PGCD de deux entiers relatifs est égal au PGCD de leurs valeurs absolues : de ce fait, on se restreint dans cette section aux entiers positifs. L'algorithme part du constat suivant : le PGCD de deux nombres n'est pas changé si on remplace le plus grand d'entre eux par leur différence. (…) La version originale de l'algorithme d'Euclide, où l'on effectue que des différences successives, est »

```
fonction euclide(a, b)
    tant que a ≠ b
        si a > b alors
           a := a − b
        sinon
           b := b − a
    retourner a
```

3. Prouver la correction et la terminaison de pgcd.