TP 1

Ce TP est à déposer à la fin de cette séance sur Moodle, en utilisant le choix « Export session as zip file » du menu « File ».
Chacun d'entre vous doit faire ce dépôt (soit deux dépôts par binôme).

L'archive `TP1.zip` à récupérer depuis Moodle contient le fichier à compléter `tp1.mlw`.

Exercice 0 : prise en main de why3

Why3 s'exécute depuis un terminal en lançant la commande suivante :

```
why3 ide tp1.mlw &
```

La première fois que vous exécutez `why3`, un message d'erreur apparaît. Vous devez d'abord exécuter la commande suivante qui permet à Why3 de savoir quels sont les démonstrateurs automatiques à sa disposition. Dans les TP, nous utiliserons les démonstrateurs Alt-Ergo, CVC4 et Z3.

```
why3 config detect
```

Vérifier que vous avez compris le fonctionnement de Why3 en effectuant les preuves du module `Ex0Test` fourni. En particulier, avant de demander à Why3 de prouver le programme, vous utiliserez la stratégie « Split VC» de Why3 (dont le raccourci clavier est la touche « s ») pour observer les différentes obligations de preuve générées par Why3.

Rajouter un cas de test dans le module de test `Ex0Test` et prouvez le depuis Why3.

Exercice 1 : maximum de deux entiers

1. Dans le fichier `tp1.mlw`, le module Ex1 spécifie de deux manières différentes le calcul du maximum de deux entiers. Pour chaque sous-programme, ajouter dans le module de test `Ex1Test`, un cas de test pour lequel les deux paramètres ont la même valeur, et tester les sous-programmes. Que se passe-t-il si la ligne suivante est enlevée de la spécification ?
   ```
   ensures { i <= result /\ j <= result }
   ```
2. Sans modifier leur spécification, programmer ces deux sous-programmes de la même manière : les deux sous-programmes retourneront le (vrai) maximum des deux entiers pris en paramètres.
3. Vérifier avec Why3 que ces deux sous-programmes respectent leur spécification. Que se passe-t-il si la ligne suivante est enlevée de la spécification ?
   ```
   ensures { i <= result /\ j <= result }
   ```
4. Ajouter dans le module `Ex1` un sous-programme `min` (avec sa spécification) calculant le minimum de deux entiers et vérifier avec Why3 que ce sous-programme est correct.

Dans ce premier exemple très simple, le code ressemble beaucoup à sa spécification. Cela ne sera bien sûr plus le cas pour les programmes que nous verrons dans les prochaines semaines.

Exercice 2 : valeur absolue d'un entier

Dans cet exercice, vous devrez écrire différentes variantes de spécification et de code du sous-programme `abs` dont un squelette est fourni dans le module `Ex2`. Il vous est demandé de donner un nom différent à ces variantes.

1. Compléter le module `Ex2` en spécifiant un sous-programme `abs_q1` (sans écrire son code) calculant la valeur absolue d'un entier. Cette spécification sera volontairement incomplète et exprimera seulement qu'`abs_q1` renvoie un entier positif ou nul.
2. Utiliser le module de test `Ex2Test` pour tester la spécification précédente. Rajouter des cas de test et tester votre spécification à l'aide de Why3. Que constatez-vous ?
3. Programmer `abs_q1` avec un calcul de valeur absolue correct et vérifier avec Why3 que ce sous-programme respecte sa spécification. Qu'en déduisez-vous ?
4. En repartant de la spécification d'`abs_q1`, spécifier un sous-programme `abs_q4`, en considérant désormais que ce sous-programme ne s'applique qu'à des entiers strictement négatifs. Programmer (donner son code) `abs_q4` en conséquence, et vérifier avec Why3 que ce sous-programme respecte sa spécification.
5. Programmer un sous-programme `abs_q5`, qui ajoute 2 à son paramètre. Spécifier `abs_q5` en prenant la même post-condition que `abs_q4`. Déterminer la pré-condition nécessaire pour que ce sous-programme respecte sa spécification. Vérifier le programme avec Why3.
6. Programmer un sous-programme `abs_q6`, qui ajoute 1 à son paramètre. Spécifier ce sous-programme de la même manière que `abs_q5`. Que se passe-t-il lorsque vous demandez à Why3 de prouver ce sous-programme ?
7. Programmer un sous-programme `abs_q7` qui calcule la valeur absolue de son paramètre. Spécifier de manière correcte et précise ce sous-programme. Vérifier que cette nouvelle spécification permet de prouver les tests du module de test.

Exercice 3 : boucle

Soit l'exemple de boucle du module `Ex3` vu en cours.

1. Modifier l'invariant de boucle en `(0 <= i <= n+2)`. Que se passe-t-il quand Why3 vérifie cet exemple ? Qu'est-ce que Why3 parvient à vérifier ?
   Dans cette question et la suivante, vous utiliserez la stratégie « Split VC » de Why3 qui permet de découper une obligation de preuve en propriétés élémentaires.
2. Revenir à l'invariant initial `(0 <= i <= n)` et modifier la condition d'entrée dans la boucle en l'expression (`not (i = n)`).
   Why3 parvient-il à vérifier l'invariant de boucle ? Pourquoi ?
   Le variant de la boucle doit-il être modifié ? Pourquoi ?
3. Écrire le code du sous-programme `boucle_q3` spécifié ci-dessous, qui décrémente un entier strictement positif jusqu'à la valeur zéro.

```
val boucle_q3 (n : int) : int
requires { 0 < n}
ensures { result = 0 }
```

4. Vérifier avec Why3 que ce programme respecte sa spécification.