Cartes à Puces
Méthodes Formelles

Exercice 1.

On considère le code source Java suivant, annoté par une spécification Esc/Java. Deux passages de code sont cachés: il ont été remplacés par les expressions `**EXPR1**` et `**EXPR2**`.

```
//@ requires t1 != null && t2 != null;
//@ ensures \result == true <==> (\forall int j; j>=0 & j < t1.length ==> **EXPR1**);
public static boolean compare(int[] t1, int[] t2){
      int i = 0;

      //@ loop_invariant i>=0;
      //@ loop_invariant **EXPR2**
      while (i < t1.length){
            if (t2[i] > t1[i])
                  return false;
            i++;
      }
      return true;
}
```

1. Que fait la fonction `compare(int[]t1,int[]t2)`?

2. La vérification par Esc/Java donne l'avertissement suivant:

```
------------------------------------------------------------------------
Tableaux.java: Warning: Array index possibly too large (IndexTooBig)
if (t2[i] > t1[i])
      ^
```

Que proposez vous pour remédier à cela ?

3. Donnez les expressions`**EXPR1**`et`**EXPR2**`.

4. Les deux fonctions suivantes sont décrites de façon informelle. Pour chacune d'entre elles, rédigez une spécification JML ainsi que le code java adéquat également annoté en JML de telle sorte qu'il puisse être vérifié par Esc/Java. Vous n'utiliserez pas de fonction de l'API java.

(a) `public static int maximum(int[] t);`
dont la fonction consiste à calculer le maximum d'un tableau d'entiers.

(b) `public static void somme(int[] t1, int[] t2, int[] t3);`
dont la fonction consiste à calculer la somme terme-à-terme des deux tableaux d'entiers `t1` et `t2`, et de placer le résultat dans `t3`.

Exercice 2.

On rappelle la spécification EscJava du tri à bulle :

```
public class TriBulle {

    /*****************************************************/
    /*@ requires t != null;                              */
    /*@ requires 0<=i && i<t.length;                     */
    /*@ requires 0<=j && j<t.length;                     */
    /*@ modifies t[i],t[j];                              */
    /*@ ensures t[i] == \old(t[j]) && t[j] == \old(t[i]); */
    /*****************************************************/
    static void echange(int[] t, int i, int j) {
         int c;
         c = t[i];
         t[i] = t[j];
         t[j] = c;
    }

    /*************************************************************************/
    /*@ requires t != null                                                   */
    /*@ requires t.length > 2                                                */
    /*@ modifies t[*]                                                        */
    /*@ ensures (\forall int i,j; 0<=j && j<i && i<t.length; t[j] <= t[i])    */
    /*************************************************************************/
    static void tri_bulle(int[] t) {
         int k,l;

         //@ loop_invariant 0 <= k && k <= t.length-1
         //@ loop_invariant (\forall int i,j; 0<=j && j<=i && i<=k; t[j] <= t[i])
         //@ loop_invariant (\forall int i; k!=0 && k<=i && i<t.length ==> t[k-1] <= t[i])
         for(k=0; k<t.length-1; k++) {
                 //@ loop_invariant l>=k && l<t.length
                 //@ loop_invariant (\forall int i; l<i && i<t.length ==> t[l] <= t[i])
                 //@ loop_invariant (\forall int i,j; 0<=j && j<=i && i<=k; t[j] <= t[i])
                 //@ loop_invariant (\forall int i,j; 0<=j && j<k && i>=k && i<t.length ==> t[j] <= t[i])
                 for(l=t.length-1; l > k; l--)
                         if (t[l-1] > t[l])
                                 echange(t,l-1,l);
         }
    }
}
```

1. Dans la spécification de la fonction echange, expliquez la ligne suivante :

```
/*@ requires 0<=i && i<t.length;   */
```

Est-ce une pré-condition ou bien une post-condition ? A quoi sert-elle ?

2. En fait, le comportement de la fonction tri_bulle n'est pas totalement spécifié. Indiquez pourquoi.

Exercice 3.

On considère un système modélisé sous smv de la façon suivante :

```
-------------------------------------------------------------------------
MODULE thermostat
VAR
       mesure : { froid, normal};
ASSIGN
       init(mesure) := { froid, normal };
       next(mesure) := { froid, normal };
-------------------------------------------------------------------------
MODULE systeme(thermostat)
VAR
  etat : { attente, initialisation_chaudiere, marche };
ASSIGN
  init(etat) := attente;
  next(etat) :=
    case
     (etat = attente):
        case
         (thermostat.mesure = froid) : initialisation_chaudiere;
         (thermostat.mesure = normal) : attente;
        esac;
     (etat = initialisation_chaudiere) : marche;
     (etat = marche):
        case
         (thermostat.mesure = froid) : marche;
         (thermostat.mesure = normal) : attente;
    esac;
    esac;
-------------------------------------------------------------------------
MODULE main
VAR
t : thermostat;
s : systeme(t);
-------------------------------------------------------------------------
```

1. Expliquez la ligne de code suivante :

```
…
init(mesure) := { froid, normal };
…
```

2. Rédigez une spécification indiquant que l'état `marche` du système succède toujours à l'état `initialisation_chaudiere`.

3. Expliquez la signification de la spécification suivante; puis indiquez en justifiant votre réponse si le système la satisfait.

```
EG EX EX EX s.etat = marche
```