Test

Exercice 1 Une fonction prend comme paramètres trois flottants représentant les longueurs des côtés d'un triangle. Elle renvoie ensuite une valeur entière indiquant s'il s'agit d'un triangle *scalène* (valeur 0), *isocèle* (valeur 1) ou *équilatéral* (valeur 2).
1.1. Produire une suite de tests pour ce programme : cas de test (Test Case), données de test (Test Data) et oracle.
1.2. Avez-vous fait seulement des tests fonctionnels ou aussi de robustesse ?

Exercice 2 Soit la spécification suivante :

```java
public static int search(List<?> list, Object element)
    // Effects: if list or element is null throw NullPointerException
    // if element is in list, returns the indice of one of its positions else -1
```

On considère la partition suivante basée sur la place de l'élément `element` dans la liste `list` :

— `element` est en début de `list`

— `element` est en fin de `list`

— `element` est à une position autre que début ou fin de `list`

2.1. Cette partition est-elle basée sur l'interface ou sur la sémantique de la fonction ?
2.2. Montrer que cette partition n'est pas disjointe.
2.3. Montrer que cette partition n'est pas complète.
2.4. Proposer une nouvelle partition disjointe et complète.

Exercice 3 On considère la fonction `mystere` suivante :

```java
double mystere(double x, int n) {
        if (n < 0) {
                x = 1 / x;
                n = -n;
        }
        double p = 1;
        while (n > 0) {
                int r = n % 2;
                if (r == 1) {
                        p = p * x;
                }
                x = x * x;
                n = n / 2;
        }
        return p;
}
```

3.1. Dessiner le graphe de contrôle correspondant aux instructions de cette fonction.
Indiquer les instructions qui peuvent engendrer des branchements.
3.2. Identifier les définitions des variables et leur portée.
3.3. Combien y a-t-il de chemins (en considérant qu'on passe au plus une fois dans la boucle) ?
3.4. Compléter le tableau suivant en indiquant pour chaque variable (en ligne) l'ensemble de ses définitions (*def*) et l'ensemble de ses utilisations en distinguant les utilisations dans des calculs (*c-use*, *computation*) et les utilisations dans des conditions (*p-use*, *predicate*). On donnera aussi l'ensemble des paires *def-use* pour chaque variable.

variable	*def*	*c-use*	*p-use*	paires *def-use*
x				
n				
p				
r				

3.5. Est-ce que le test `mystere(2, -1)` couvre toutes les instructions ? La réponse doit être argumentée. Dans la négative, on indiquera quels tests ajouter pour toutes les couvrir.
3.6. Est-ce que le test `mystere(2, -1)` couvre toutes les décisions ? La réponse doit être argumentée. Dans la négative, on indiquera quels tests ajouter pour toutes les couvrir.

Exercice 4 Soit le programme ci-dessous :

```
1  void foo(boolean a, boolean b, boolean c) {
2      if (a || (b && c)) {
3          out.println("ok");
4      }
5      out.println("fin");
6  }
```

4.1. Donner les éléments à couvrir pour chacun des critères suivants : instructions (I), décisions (D), conditions (C), décisions/conditions (DC), conditions multiples (MC), MC/DC (Modified Condition/Decision Condition).
4.2. Donner les jeux de tests du programme couvrant les critères et illustrer qu'ils sont différents.

Exercice 5 Considérons l'algorithme de l'exercice 3.
5.1. Donner les éléments à couvrir pour chacun des éléments suivants : *all-defs*, *all-use*, *all-p-use*, *all-c-use*, *all-def-use-paths*.
5.2. Donner des jeux de tests couvrant les critères et illustrer qu'ils sont différents.

Exercice 6 Soit la méthode Java du listing 1.
6.1. Donner son graphe de contrôle.
6.2. Donner une suite de tests TS_n qui couvre tous les nœuds du graphe de contrôle.

```java
// Outputs result = 0 + 1 + 2 + ... + |value|
// if results > maxInt the error
static int maxSum(int maxInt, int value) {
    int result = 0;
    int i = 0;
    if (value < 0) {
        value = - value;
    }
    while (i < value && result <= maxInt) {
        i++;
        result = result + i;
    }
    if (result <= maxInt) {
        return result;
    } else {
        throw new RuntimeException("error");
    }
}
```

Listing 1: La méthode Java maxSum

6.3. La suite TS_n couvre-t-elle tous les arcs ? Si oui, indiquer les données de tests qui effectuent la couverture des arcs. Sinon, ajouter des données de test pour obtenir une suite de tests TS_a qui couvre tous les arcs.
6.4. Indiquer quelles sont les lignes de code correspondant aux définitions de la variable `result` (ensemble *defs(`result`)*). Même question pour les ensembles d'utilisation en calcul *c-use(`result`)* et d'utilisation en prédicats *p-use(`result`)*.
6.5. Donner une suite de tests TS_d qui couvre le critères *all-p-uses* pour `result`.