Examen de test et vérification

Exercice *Tests de Palindrome* (10 points)

Un palindrome est une figure de style désignant un mot ou un texte dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche. Par exemple, "kayak" ou encore "Esope reste ici et se repose" sont des palindromes. Dans cet exercice, nous ne considérons pas les caractères accentués.

Soit la fonction suivante, écrite en Java, permettant de déterminer si la chaîne de caractères passée en paramètre est un palindrome :

```
1  public static boolean palindrome(String s) {
2    if (s == null) {
3      return false;
4    } else {
5      int low = 0;
6      int high = s.length() - 1;
7      while (low < high && s.charAt(low) == s.charAt(high)) {
8        low++;
9        high--;
10     }
11     return low == high;
12   }
13 }
```

1. Dessinez le graphe de flot de contrôle correspondant à la fonction *palindrome*, en décomposant les conditionnelles. (1pt)
2. Déterminez les cas de test (*i.e.*, données de test + résultat attendu) minimaux pour couvrir toutes les instructions (*i.e.*, couverture de tous les nœuds du graphe). (1pt)
3. Si ces cas de test ne permettent pas la couverture de tous les arcs du graphe, ajoutez-en de nouveaux pour couvrir tous les arcs. (0,5pt)
4. Si besoin, ajoutez des cas de test qui permettent : (1pt)
   — D'exécuter la fonction sans passer dans la boucle `while`,
   — D'exécuter la fonction en passant exactement une fois dans la boucle `while`,
   — D'exécuter la fonction en passant au moins deux fois dans la boucle `while`.
5. Si besoin, ajoutez des cas de test pertinents qui manqueraient. (1pt)
6. Écrivez, dans une classe de test JUnit (en précisant si vous utilisez JUnit 3.8 ou 4), des méthodes de test pour tester la fonction *palindrome* pour tous les cas de test que vous avez identifiés. (1pt)
7. Modifiez la fonction *palindrome* de sorte qu'elle lève une exception (à définir) dans le cas où la chaîne `s`, passée en paramètre, vaut `null`, et modifiez également le test correspondant. (0,5pt)
8. Certains de vos tests ne devraient pas passer, car la fonction *palindrome* est erronée. Identifiez la faute. (1pt)
9. Si possible, identifiez un cas de test qui n'exécute pas la faute. (0,5pt)

10. Si possible, identifiez un cas de test qui exécute la faute, mais n'aboutit pas à un état d'erreur. (0,5pt)
11. Si possible, identifiez un cas de test qui entraîne une erreur, mais pas une défaillance. (0,5pt)
12. Identifiez un cas de test qui montre une défaillance. (0,5pt)
13. Réparez la faute et vérifiez que l'ensemble de vos tests passe. (1pt)

Exercice *Vérification de Palindrome* (10 points)

Afin de pouvoir utiliser l'outils Why pour la vérification du programme précédant, nous transformons la chaîne de caractère (`String`) en entrée, par un tableau de booléens (`boolean[]`). En effet, Why ne traite pas les appels de méthodes telles que `length()` et `charAt(i)`. Notre fonction palindrome permet donc de vérifier si un nombre binaire (représenté par un tableau de booléen) peut se lire dans les deux sens (par exemple 010, 10011001, 1, ...). Nous supprimons également le test de nullité du tableau qui devra être intégré dans la précondition. Nous obtenons donc le code suivant :

```
1  public class Verif2015 {
2  /*@
3    @ requires ..(1)..;
4    @ ensures \result <==> \forall integer i; ..(2).. ;
5  */
6
7  public static boolean palindrome(boolean[] s) {
8      int low = 0;
9      int high = s.length - 1;
10
11 /*@ loop_invariant
12   @ 0 <= low && high < s.length &&
13   @ high + low == s.length -1 &&
14   @ \forall integer i; ..(3).. ;
15 */
16     while (low < high && s[low] == s[high]) {
17       low++;
18       high--;
19     }
20     return low == high;
21 }
```

1. Complétez la précondition ..(1).. . (1pt)
2. Complétez la postcondition ..(2).. . (2pt)
3. Complétez l'invariant de boucle ..(3).. . (3pt)
4. Ajoutez un variant de boucle. (1pt)
5. Utilisez le logiciel Why pour démontrer que le programme ci dessus est faux et corrigez l'erreur pour qu'il soit vérifiable. (1pt)
6. Supprimez la première ligne de l'invariant et expliquez l'erreur produite par Why. (1pt)
7. Supprimez la seconde ligne de l'invariant et expliquez l'erreur produite par Why. (1pt)