Harjoitus 4. 18.-19.4. / Exercise 4

Tehtävät 3., 4. ja 5. koskevat funktiota splitInTwo3(), jonka koodi on dokumentin lopussa. Funktio on korjattu versio 3. harjoituksen splitInTwo() -funktiosta. The code for the splitInTwo3() function (for the tasks 3., 4., and 5.) is included at the end of this document:

1. Muunna kuvan "Figure 1 - A seminar during registration" tilakaavio (state diagram) sivulta http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm tilasiirtymäpuuksi. Virheellisten/odottamattomien syötteiden testejä vastaavia oksia ei tarvitse piirtää tilasiirtymäpuuhun. Kuinka monta testitapausta tarvitaan tila- ja siirtymäkattavuuden saavuttamiseksi (virheellisten syötteiden testejä ei tarvitse laskea mukaan)?

Transform the state machine in "Figure 1 - A seminar during registration" on page http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm into a transition tree. You don’t have to add the branches for invalid inputs. How many test cases are needed to reach state and transition coverage?
The tree has 12 leaves, so 12 test cases are needed to achieve state and transition coverage. Note 1: the tree seems to have duplicate branches with same input and target state (two transitions from "full" to "full" with same input event "student dropped"). However, the transitions have different guards (conditions) that need to be satisfied (they are mutually exclusive). So they represent different behaviors that need to be shown in the tree and tested: the test cases will have different sequences of inputs that lead to the different behaviors.

Note 2: State “full” has an internal transition for the input event “Enroll student”. This is not shown by a state transition arrow from the state back to itself (unlike the transitions for "student dropped" input event). However, the transition needs to be shown in the tree, because testing it will need its own test case. But, there seems to be a problem or error in the state chart because on other transitions the input event “Student enrolled” is used. Is “enroll student” the same input event or a different event? This needs to be checked from the person who has specified the state chart. Anyway, state “full” and the transitions from/to the state and their guards need to be carefully studied to be able to form the needed test cases that will exercise the transitions. Maybe some other problems will also be discovered. However, it is not the job of the test case designer to correct them, but to report them as defects of the specification.

2. Tutustu Skenaariotestaukseen (Scenario Testing) tällä sivulla http://testingeducation.org/BBST/scenario/. Miten skenaariotestaus eroaa käyttötapauksista?

Introduce yourself to Scenario Testing on this page http://testingeducation.org/BBST/scenario/. How does Scenario Testing differ from Use Case Testing?

Scenarios represent a possible course of events and actions of the user of the system that lead to complex situations in terms of activities and/or data. These kind of “stories” can be used to discover errors that use case based tests could not find. Use cases represent a “modularized” specification of the behavior of the system and they do not cover long sequences of actions and complex conditions like scenarios do. The story –like form of scenarios can also help convince management and even the customer of the importance of correcting the problems uncovered by executing a scenario.

3. Laadi funktion splitInTwo3() kontrollivuoverkko. Huomaa, että verkkoon tulee useita 'exit'-solmuja, joista ei lähde kaaria muihin solmuihin (jokaisista 'return' -lausetta vastaa oma 'exit'-solmu).

Construct the control flow graph for the splitInTwo3() function. Note that there are several exit nodes (for each return statement) in the graph that do not have any arcs leaving them.

A note on drawing CFG’s: the course book uses two different ways for drawing IF-statements in control flow graphs as shown by the picture below. In the graph fragment a) there is the extra node marked as “D” that does not represent any statements. It represents the end of the IF-construct (the ENDIF) and the amalgamation of the two control flows branching out from the test of the condition of the IF statement (node “A”). See the code on page 153 and the CFG on page 154 for a concrete example (also the CFG on p. 144 and its explanation on p. 145). In some other examples the book shows IF statements drawn as in the fragment b) where there is no node representing the ENDIF, but the two control flow paths simply coalesce at the next statement after the IF statement (node “E” in both fragments). I recommend using way b) since it leads to a simpler graph with fewer nodes.
The CFG for function splitInTwo3():

```
S -> A
  |
  v
A -> B
  |
  v
B -> C
  |
  v
C
```

```
S -> A
  |
  v
A -> B
  |
  v
B -> C
  |
  v
C
```

```
S -> A
  |
  v
A -> B
  |
  v
B -> C
  |
  v
C
```

```
S -> A
  |
  v
A -> B
  |
  v
B -> C
  |
  v
C
```

```
S -> A
  |
  v
A -> B
  |
  v
B -> C
  |
  v
C
```

```
S -> A
  |
  v
A -> B
  |
  v
B -> C
  |
  v
C
```

```
S -> A
  |
  v
A -> B
  |
  v
B -> C
  |
  v
C
```
The numbered nodes represent branching points (testing of the condition of an IF or a WHILE statement) and the others represent groups of (normal) statements executed sequentially.

4. Kuinka monta testitapausta vähintään tarvitaan tehtävän 3 kontrollivuoverkon perusteella
   a) vain lausekattavuuden
   b) haarautumakattavuuden

saavuttamiseen?

Based on the CFG constructed in task 3, how many test cases are at least needed to achieve
   a) statement coverage only?
   b) branch coverage?

At the minimum, five test cases are needed to reach statement (C0) and branch (C1) coverage. It is possible to reach both C0 and C1 coverage with the same test cases that execute the following paths in the CFG:

i. 1, A
ii. 1, B, 2, C
iii. 1, B, 2, D, 3, E, 4, G, 3, 5, K, L
iv. 1, B, 2, D, 3, E, 4, F, G, 3, 5, H, 6, J, L
v. 1, B, 2, D, 3, E, 4, F, G, 3, 5, H, 6, I, L

Note that the choice for which branch to take at node 5 (an IF) depends on the execution of the WHILE-loop starting at node 3, because variable found may be assigned a value there. For example, the path (1, B, 2, D, 3, E, 4, G, 3, 5, H, 6, J, L) is not possible to achieve because found would have the value false and the condition of the IF at 5 would evaluate to false leading to the execution of statement K next.

5. Suunnittele konkreettiset testitapaukset (vähimmäismäärä) syötteineen, joilla saavutetaan lause- ja haarautumakattavuus splitInTwo3() -funktiossa.

Design the concrete test cases (minimum number) with their inputs to achieve statement and branch coverage for splitInTwo3().

For example, test cases corresponding to the paths above in task 4.: 

i. Input: original = null, marker = ‘n’ (any valid value)  
   Output: null

ii. Input: original = “” (empty string), marker = ‘n’ (any valid value)  
   Output: result[0] = “”, result[1] = “”

iii. Input: original = "m", marker = ‘n’ (marker not found in original) 
   Output: result[0] = “m”, result[1] = “”

iv. Input: original = "nk", marker = ‘n’ (marker found, not the last char of original) 
   Output: result[0] = “”, result[1] = “k”

v. Input: original = "n", marker = ‘n’ (marker found at the end of original) 
   Output: result[0] = “”, result[1] = “”
Determine the Equivalence Classes and the Boundary Values for the following function:

**Function**

```plaintext
BMI(weight, height) calculates a person's Body Mass Index using the formula:

\[
\text{weight} / (\text{height} \times \text{height})
\]

Weight is given in kilograms (an integer) and height in meters (a real number) with the precision of two decimals. If there are more decimals, the input is rounded to right precision before calculation. The function returns the BMI as a real number with one decimal precision.

The weight and the height must both be positive numbers. Otherwise, the function returns -1.0.
```

```java
public String[] splitInTwo3(String original, char marker) {
    int i, pos, len;
    boolean found;
    char ch;
    String[] result = null;
    if (original == null) {
        return null;
    }
    result = new String[2];
    len = original.length();
    if (len == 0) {
        result[0] = "";
        result[1] = "";
        return result;
    }
    i = 0;
    pos = -1;
    found = false;
    while (!found && i < len) {
        ch = original.charAt(i);
        if (ch == marker) {
            pos = i;
            found = true;
        }
        i += 1;
    }
    if (found) {
        result[0] = original.substring(0, pos);
        if (pos == len-1) {
            result[1] = "";
        } else {
            result[1] = original.substring(pos + 1);
        }
    } else {
        result[0] = original;
        result[1] = "";
    }
    return result;
}
```