Testing und Debugging: Übung 01

Abgabetermin Siehe Canvas Kurs

1 Assertions in anderen Programmiersprachen

In der Vorlesung haben Sie gesehen, wie Assertions in Python benutzt werden. Andere Programmiersprachen kennen dieses Konzept ebenfalls. Allerdings hat jede Sprache ihre Eigenheiten, wie Assertions in ihr verwendet werden.

1.1 Assertions in Java (10 Punkte)

Gegeben sei der folgender Code:

Assertion in Java (AssertKeyword.java)

```
public class AssertKeyword
{
  public static double subAndSqrt( double a, double b )
  {
    double result = Math.sqrt( a - b );

    assert ! Double.isNaN( result ) : "Berechnungsergebnis ist NaN!";

    return result;
  }

  public static void main( String[] args )
  {
    System.out.println( "Sqrt(10-2)=" + subAndSqrt(10, 2) );
    System.out.println( "Sqrt(2-10)=" + subAndSqrt(2, 10) );
  }
}
```

Ausgeführt wird dieser Code mit folgendem Kommando (es liegt bereits eine vorkompilierte Version des Programms vor):

```
java AssertKeyword
```

Aufgaben

1. *Welches Ergebnis **erwarten** Sie? (1 Punkt)*

   ```
   .
   .
   ```

2. *Welches Ergebnis **bekommen** Sie? (1 Punkt)*

   ```
   .
   .
   ```

3. *Wieso wird kein AssertionError geworfen? (4 Punkte)*

```
.
.
```

4. *Wie sorgt man dafür, dass bei Java das* `assert` *Statement angewendet wird? (4 Punkte)*

```
.
.
```

1.2 Assertions in C (30 Punkte)

In C werden Assertions in der Header Datei *assert.h* (Linux Pfad */usr/lib/assert.h*) als Precompiler Makro `assert(EXPRESSION)` definiert. Im Vergleich zu Java oder Python gibt es in C bei der Benutzung des `assert` Makros keine (direkte) Möglichkeit einen zusätzlichen Text mit anzugeben. Man kann aber ausnutzen, dass ein nicht-leerer String in einer boolschen Verknüpfung als wahr interpretiert wird.

Wie Assertions in C benutzt werden zeigen die Zeilen 3 (include des Headers) und 12 (Benutzung des `assert` Makros inkl. Text) in untenstehendem Quellcode von *mysqrt.c*. Beim Compilieren sind für die Benutzung von Assertions keine zusätzlichen Schalter notwendig.

Assertion in C (mysqrt.c)

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <assert.h>
4
5  double mysqrt(double x) {
6
7    /* Vorbedingung */
8
9    double result = sqrt(x);
10
11   /* Nachbedingung */
12   assert((result * result == x) && \
13          "Das Quadrat des Ergebnisses ist gleich dem Eingabewert.");
14
15   return result;
16 }
17
18 int main() {
19   double input = 2.44140625;
20   double output;
21   int i;
22
23   for (i = 0; i < 10; i++)
24   {
25     output = mysqrt(input);
26     printf("Die Wurzel aus %.16f ist %.16f.\n", input, output);
27     input = output;
28   }
29   return 0;
30 }
```

Obige Datei kann mit folgenden Kommandos compiliert und aufgerufen werden:

```
gcc mysqrt.c -lm -o mysqrt
./mysqrt
```

Tipp
Der Schalter `-lm` ist notwendig, da die Funktion `sqrt` in der Bibliothek *libm.so* implementiert und in der Header Datei *math.h* definiert ist.

Nach dem Aufruf von `./mysqrt` erhalten Sie folgende Ausgabe:

```
Die Wurzel aus 2.4414062500000000 ist 1.5625000000000000.
Die Wurzel aus 1.5625000000000000 ist 1.2500000000000000.
mysqrt: mysqrt.c:13: mysqrt: Assertion `result * result == x && "Das Quadrat
des Ergebnisses ist gleich dem Eingabewert."' failed.
```

Aufgaben

1. *Warum wird bei der dritten Ausführung der Funktion `mysqrt` eine Assertion ausgelöst? (5 Punkte)*

   .
   .

2. *Schreiben Sie das Assert Statement aus Zeile 12 so um, dass es eine **sinnvolle** Verbesserung des gegeben Statements ist; d.h. es soll weiterhin das Resultat von `sqrt` auf Korrektheit hin prüfen. Das Programm soll mit dem veränderten Statement ohne Fehler oder Assertion durchlaufen können. (15 Punkte)*

   .
   .
   .

3. *Was wäre ein **sinnvolles** Assert Statement für die Vorbedingung in der `mysqrt` Funktion? Geben Sie das Statement so an, dass es direkt in den Quellcode in Zeile 8 eingetragen werden kann. (5 Punkte)*

   .
   .

4. *Wie kann man in C die Ausführung von Assertions in einem Programm verhindern, **ohne alle Assert Statements aus dem Quellcode zu entfernen**? (5 Punkte)*

   .
   .

2 Allgemeines zu Assertions (10 Punkte)

Gegeben seien die folgenden drei syntaktisch korrekten Assert Statements (Python-Style):

```
1 assert setValue(10), "Checke, ob der Wert 10 gesetzt werden kann"
2 assert isinstance(x, int), "Variable x ist ein Integer"
3 assert 2 == 1 + 1, "Checke, ob Python richtig addiert"
```

Aufgaben

1. *Wie schaltet man bei Python die Ausführung der Assert Statements ab? (3 Punkt)*

   .
   .

2. *Welches der obigen Statements (Zeilennummer) kann beim Abschalten der Assertions für Probleme sorgen? Begründen Sie Ihre Antwort. (5 Punkte)*

   .
   .
   .

3. *Welches weitere der obigen Statements ist ebenfalls nicht sinnvoll? Begründen Sie Ihre Antwort kurz. (2 Punkte)*

   .
   .

3 Programmieraufgabe: Blackbox-Testing (100 Punkte)

Beim Testen von Software kommt es mit unter vor, dass Sie mit dem Testen eines Moduls oder einer Bibliothek beauftragt werden, von der Ihnen lediglich die Spezifikation und die Schnittstellenbeschreibung zur Verfügung steht.

EIN GROBER LEITFADEN IN EINER SOLCHEN SITUATION IST:

1. Lesen Sie die Spezifikation und Schnittstellenbeschreibung genau durch.
2. Lesen Sie die Spezifikation und Schnittstellenbeschreibung noch einmal genau durch. [1]
3. Verwenden Sie in Ihren Tests **alle** beschrieben Funktionen in einer sinnvollen Reihenfolge. [2]
4. Verwenden Sie verschiedene Werte für die Parameter der einzelnen Klassen und Funktionen.
5. Testen Sie sinnvolle Extremwerte für Parameter (minimale, maximale oder speziell exponierte Werte im Definitionsbereich).
6. Testen Sie Funktionen mehrfach (vielleicht tritt eine spezielle Situation erst nach mehrmaligem Aufruf ein).

[1] Dieser Schritt ist so wichtig, dass man ihn mindestens zweimal macht!
[2] z.B. Objekt erzeugen, Objekt manipulieren, Objekt zerstören.

Dateien für die Programmieraufgabe

submit.cfg	Konfigurationsdatei für das Abgabeskript
run_test.py	Datei testet Ihre Implementierung von `testQueue()` in *uebung01.py*
Grading/*init*py	Hinweisdatei für Python: Das Verzeichnis ist ein Modul
Grading/Grading.py	Quellcode der Testklasse, die in *run_test.py* verwendet wird
MyQueue.pyc	Bytecode Datei der fehlerhaften Warteschlangen
uebung01.py *	In dieser Datei müssen Sie die Funktion `testQueue()` gemäß Aufgabenstellung implementieren

Die mit [*] markierten Dateien müssen **bearbeitet** und **abgegeben** werden.

In dieser Programmieraufgabe sollen mehrere, zum Teil fehlerhafte, Implementationen der, aus der Vorlesung bekannten Warteschlangenklasse (`MyQueue`) getestet werden. Sie haben für Ihre Testfunktion lediglich die unten stehende Spezifikation zur Verfügung — Einblick in den Quelltext der MyQueue Klassen haben Sie nicht.

Spezifikation: Die MyQueue Klasse stellt eine FIFO Warteschlange mit fester Länge für Integers bereit.

Der Konstruktor hat einen Parameter: ein Integer > 0 der die maximale Anzahl von Elementen angibt, die die Warteschlange speichern kann.

`empty()` gibt genau dann **True** zurück, wenn die Warteschlange keine Elemente enthält.

`full()` gibt genau dann **True** zurück, wenn die Warteschlange kein weiteres Element speichern kann.

`enqueue(i)` versucht den Integer Wert `i` in der Warteschlange abzulegen. Die Funktion gibt **True** zurück, falls dieser Versuch erfolgreich war und **False** wenn die Warteschlange voll ist.

`dequeue()` entfernt (nach dem FIFO Prinzip) einen Wert aus der Warteschlange und gibt diesen zurück. Ist die Warteschlange leer, so wird **None** zurückgegeben.

Beispiel:

```
q = MyQueue(1)
is_empty = q.empty()
success = q.enqueue(42)
is_full = q.full()
value = q.dequeue()
assert value == 42, "Die MyQueue kennt die Antwort..."
```

Ihre Aufgabe

Ihre Aufgabe ist es, die Funktion `testQueue()` in der Datei *uebung01.py* so mit Funktionsaufrufen der MyQueue-Klasse und `assert` Statements zu ergänzen, dass Sie alle 8 Fehler finden.

Wenn Sie mit `python3 uebung01.py`[a] das Python Skript ausführen, dann wird Ihre Implementation zunächst gegen eine fehlerfreie Implementation der MyQueue Klasse getestet. Dieser Aufruf sollte lediglich folgende Ausgabe erzeugen (wobei natürlich Ihre Gruppennummer und Name und zugehörige Matrikelnummer Ihrer Gruppenmitglieder angezeigt werden muss); es dürfen **keine AssertionErrors** erscheinen:

```
====================
Aufgabe:       01
Gruppennummer: 99

Matrikelnummer, Name
--------------------
        99999, Sebastian Stigler
====================
```

Das Skript *run_test.py* sorgt dafür, dass Ihre Testfunktion auf die 8 fehlerhaften Implementationen angewendet wird. Aufgerufen wird das Skript mit `python3 run_test.py uebung01.py`.

Dieses Skript testet im 1. Test, ob Sie Ihre Gruppennummer und die Namen und Matrikelnummern Ihrer Gruppenmitglieder richtig eingetragen sind und Sie nur dort, wo es im Quellcode angegeben ist, Änderungen vorgenommen haben. Dies ist wichtig, da ansonsten das automatische Korrektur System das Ergebnis keiner Übungsgruppe zuordnen kann (⇒ Sie erhalten überhaupt keine Punkte für die Programmieraufgabe!!).

Der zweite Test prüft, ob Sie alle Funktionen der Klasse `MyQueue` benutzt haben und ob die fehlerfreie Implementation der Klasse keine Fehler wirft. Im Anschluss werden die 8 fehlerhaften Implementationen mit Ihrer Testfunktion getestet. Ist der erste Test fehlerfrei, so erhalten Sie für diesen und jeden weiteren Test, den das Testskript als bestanden anzeigt, 10 Punkte.

[a] Es ist wichtig, das Skript mit *python3* und nicht nur mit *python* aufzurufen!

Abgabe der Programmieraufgabe

1. Falls noch nicht vorhanden, dann erstellen Sie sich auf https://github.com einen Account. (Pro Gruppe ist nur ein Account und eine Abgabe nötig!)
2. Eine Person aus der Gruppe gibt auf Canvas im *Hier geht es los!* Modul den Namen des Github Accounts an.
3. Gehen Sie in Ihrer virtuellen Maschine in das Aufgabenverzeichnis (wo sich Ihre bearbeitete Aufgabe und die Datei `submit.cfg` befinden).
4. Tippen Sie in der Konsole den Befehl *submit*. Dieser wird beim ersten Ausführen nach dem Githubtoken Ihres github Accounts fragen. Ebenfalls im *Hier geht es los!* Modul ist ein Video verlinkt (*Token für die automatische Abgabe erstellen*) in dem beschrieben wird, wie sie den Token erstelle und in der VM eintragen. Anschließend werden die bearbeiteten Abgaben verschlüsselt auf https://gist.github.com abgelegt.
5. Diese Datei wird nach dem Abgabetermin automatisch zur Korrektur heruntergeladen.

Viel Erfolg