Dr. Annette Bieniusa
Mathias Weber, M. Sc.
Peter Zeller, M. Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Lösungshinweise/-vorschläge zum Übungsblatt 15: Software-Entwicklung 1 (WS 2017/18)

Die Hinweise und Vorschläge in diesem Dokument sollen der Lösungsfindung dienen und erheben demnach weder Anspruch auf Vollständigkeit noch Korrektheit. Sollten Sie Fehler finden, würden wir uns freuen, wenn Sie uns diese mitteilen. (Kontaktdaten finden Sie auf unserer Webpräsenz.)

Aufgabe 1 Terminierung

Geben Sie für die folgenden Prozeduren jeweils eine sinnvolle Vorbedingung an, so dass die Prozedur immer ohne Fehler terminiert, wenn die Vorbedingung erfüllt ist.

Beweisen Sie dann die Terminierung der Prozedur für alle erlaubten Werte. Verwenden Sie dazu das Verfahren aus der Vorlesung.

a) Binärsuche in Java:

```
1  public class DataSet {
2    int key;
3    String data;
4
5    DataSet (int key, String data) {
6      this.key = key;
7      this.data = data;
8    }
9    /* Für bessere Übersichtlichkeit im Skript verzichten wir hier
10   auf Getter/Setter-Methoden */
11 }
12
13 public static DataSet search (int x, DataSet[] f, int lo, int hi) {
14   // Element nicht gefunden
15   if (hi <= lo) {
16     return null;
17   }
18   int mid = lo + (hi-lo) /2;
19   if (x < f[mid].key) {
20     // Suche im Intervall [lo, mid)
21     return search (x, f, lo, mid);
22   }
23   if (x > f[mid].key) {
24     // Suche im Intervall [mid+1, hi)
25     return search (x, f, mid+1, hi);
26   }
27   // Element gefunden
28   return f[mid];
29   }
```

1. **Vorbedingung:** `f != null && (hi <= lo || (lo>=0 && hi < f.length))`
2. **Nicht verlassen des Parameterbereichs:** Da `f` nicht verändert wird, ist `f` im Aufruf `search(x, f, lo, mid)` in Zeile 21 ebenfalls `!= null`. Gleiches gilt für den Aufruf in Zeile 25.

In Zeile 18 ist `lo < hi` und `hi-lo < f.length`. Somit ist `mid` zwischen lo und hi. Im Aufruf in Zeile 21 ist somit `mid < hi < f.length` und im Aufruf in Zeile 25 ist `0 <= lo <= mid < mid +1`.

3. **Abstiegsfunktion:** Als Abstiegsfunktion wählen wir
   `h(x, f, lo, hi) = |hi - lo|`
4. **Kleiner Werden der Parameter:**
   Aufruf `search(x, f, lo, mid)` Zeile 21:
   Wir wissen, dass `lo < hi` wegen der **`if`**-Bedingung in Zeile 15 und somit `|hi - lo| = hi - lo` und `hi - lo > 0`. Somit ist `mid >= lo` und `mid < hi` wegen der Rundungssemantik der Integer-Division. Zu zeigen ist, dass die Parameter in den rekursiven Aufrufen bezüglich `h` kleiner werden:
   `h(x, f, lo, hi) = |hi - lo| = hi - lo > mid - lo = |mid - lo| = h(x, f, lo, mid)`

   Aufruf `search(x, f, mid+1, hi)` Zeile 25:
   Wir wissen, dass `lo < hi` (siehe oben) und somit `|hi -lo| = hi - lo` und `hi - lo > 0`. Außerdem ist `mid >= lo` und `mid < hi` wegen der Rundungssemantik der Integer-Division und somit `mid + 1 > lo`. Somit gilt:
   `h(x, f, lo, hi) = |hi - lo| = hi - lo > hi - (mid+1) = |hi - (mid+1)| = h(x, f, mid+1, hi)`
   da `hi - (mid+1) >= 0`.

b) Hier ist `TreeNode` wie in der Vorlesung definiert. Sie können davon ausgehen, dass totale Funktionen `size(TreeNode t)` für die Anzahl der Knoten im Baum und `height(TreeNode t)` für die Höhe des Baums definiert sind.

```
public TreeNode mirror(TreeNode t) {
  if (t == null) {
    return null;
  }
  return new TreeNode(t.getMark(), mirror(t.getRight()), mirror(t.getLeft()));
}
```

1. Wir zeigen die Terminierung für alle gültigen Bäume mit `t` als Wurzelknoten, schränken also den Parameterbereich nicht weiter.
2. Damit ist der Beweis, dass der gültige Parameterbereich nicht verlassen wird, auch relativ einfach, da ein gültiger Wurzelknoten `node` einen gültigen linken und rechten Teilbaum besitzt.
3. Als Abstiegsfunktion wählen wir die Höhe des Baums:

   $$h(t) = height(t)$$

   Die Abstiegsfunktion h bildet offensichtlich in die natürlichen Zahlen ab, da die Höhe eines Baums nicht negativ sein kann.
4. Zu zeigen bleibt, dass die Parameter für rekursive Aufrufe echt kleiner werden.

   - Aufruf `mirror(node.getLeft())`:
     $h(node) = height(node) > height(node.getLeft()) = h(node.getLeft())$
     Dies gilt, da es sich bei `node.getLeft()` um eine kleinere Instanz eines Baumes handelt, da `node` kein Blatt ist (wegen der **`if`**-Bedingung).
     Formal bedeutet das, dass

     $$\begin{aligned} height(node) &= 1 + Math.max(height(node.getLeft()), height(node.getRight())) \\ &\geq 1 + height(node.getLeft()) \\ &> height(node.getLeft()) \end{aligned}$$

   - Aufruf `mirror(node.getRight())`:

Gleiche Begründung wie oben.