Übungsblatt 13

1. Gegeben sei folgende Klasse:

```
class List {
   int i;
   List next;

   // invariant   next != null  =>  i == next.i

   // i in der ganzen Liste erhöhen
   void inc() {
        …
   }
}
```

a. Schreiben Sie eine formale Spezifikation für die Methode `inc()`

b. Schreiben Sie alle Beweisverpflichtungen für `inc()` auf

c. Schreiben Sie eine rekursive Implementierung der Methode `inc()`

d. Überprüfen Sie, ob die Implementierung die Beweisverpflichtungen erfüllt

e. Falls nicht, geben Sie eine alternative Lösung an, die die Beweisverpflichtungen erfüllt

2. Gegeben seien folgende Klassen:

```
public class DLHead {

  private class DLNode {
    DLNode prev;

    DLNode next;

    Object elem;
  }

  private DLNode first;

  private DLNode last;

  public void add( Object obj ) { /* ... */ }

  public Object getFirst() { return first.elem; }

  public void removeFirst() { /* ... */ }

  ...
}
```

a. Diese Klassen sollen eine übliche doppelt-verkettete Liste beschreiben. Schreiben Sie die notwendigen Invarianten für die Klassen `DLHead` und `DLNode`.

b. Es soll zusätzlich sichergestellt werden, dass die Knoten aufsteigend nach dem Hashwert der Elemente sortiert werden.

a. Welche zusätzlichen Invarianten sind dafür notwendig?

b. Wie kann diese Eigenschaft durch die Verwendung der zusätzlichen Model-Fields `min` und `max` sichergestellt werde?
Das Model-Field `min` enthält den minimalen Hashwert der Liste, das Feld `max` enthält den maximalen Hashwert der Liste.