

Probeklausur (180 min.)

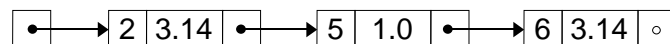
zu erreichende Mindestpunktzahl: 18 von 36

Aufgabe 1 (55 min, 11 Punkte)*Dünn besetzte Vektoren*

Ein Vektor $(a_0, a_1, \dots, a_{n-1})$, $a_i \in \mathbf{R}$, ist *dünn besetzt*, wenn seine Elemente bis auf wenige Ausnahmen gleich Null sind. Für große n hieße es Speicher verschwenden, wenn man dünn besetzte Vektoren als Felder des Typs `float []` darstellen wollte. Häufig wählt man daher eine verkettete Liste, die genau die von Null verschiedenen Elemente und dazu die jeweiligen Indexwerte enthält. Beispielsweise würde der Vektor

(0, 0, 3.14, 0, 0, 1.0, 3.14)

als veränderliches Objekt wie folgt dargestellt werden:



Beachten Sie, dass diese Repräsentation keine Auskunft über n gibt und für beliebig große Vektoren geeignet ist!

In Java können wir wie folgt formulieren:

```
interface FloatVector { // Modell: v::[float], n :: Int
    // anfangs v == replicate n 0
    // Inv.: length v == n

    public int firstIndexNeg();
        // Vor.: any (<0)v
        // Eff.: result == minimum [i|i<-[0..n-1],v!!i<0]
    .....
}

class FloatList implements FloatVector {
    class Cell {
        int index; float value;
        Cell next;
        Cell(int i, float f, cell c) {
            index = i; value = f; next = c; }
    }

    Cell first = null; // Inv.: ...
                    // Bed.: ...

    public int firstIndexNeg() {
        // Vor.: ...
    }
}
```

```

        // Eff.: ...
        .....}
.....
}

```

Gesucht ist die konkrete Invariante (4 P.), die Abstraktionsfunktion (3 P.), ferner Voraussetzung und Effekt der Implementierung von `firstIndexNeg` (4 P.), alle sowohl informell als auch formal in (Pseudo-) Haskell.

Aufgabe 2 (45 min, 9 Punkte)

Einfacher interner Iterator

Die folgende Schnittstelle für Verzeichnisse (tabellierte Abbildungen) sieht eine Traversierungsoperation über alle verzeichneten Einträge vor:

```

interface Mapping<K,D> { // Modell: m :: K->D,
                        // anfangs überall undefiniert
void iterate(Action<D> a);
    // Vor.: -
    // Eff.: Jedes x aus dem Wertebereich von m mit a.cond(x)
    //       ist durch a.func(x) ersetzt; alle andere sind
    //       unverändert.
...
}

interface Action<X> {
boolean cond(X x);
X func(X x);
}

```

a) (25 min, 5 P.) Für eine Implementierung

```
class HashedMapping<K,D> implements Mapping<K,D> ,
```

die ein offenes Streuspeicherverfahren (Hashing) mit einem Feld von „Buckets“ praktiziert, ist die Repräsentation und die zugehörige Implementierung von `iterate` gesucht. (Die Angabe von Abstraktionsfunktion, Invariante, Voraussetzung und Effekt ist *nicht* verlangt.)

b) (20 min, 4 P.) Bei den unteren Lohngruppen ($\text{Lohn} \leq 1500 \text{ €}$) der Firma Theuerkauf gibt es eine Lohnerhöhung um 8,8%, höchstens aber auf 1500 €. Die Firma führt eine Lohnliste mit Einträgen der Form (Personalnummer, Lohn). Wir modellieren die Lohnliste als ein Objekt

```
Mapping<Integer, Float> salaries;
```

Gesucht ist eine Klasse `Raise` derart, dass man mit der folgenden Anweisung die Lohnerhöhung in der Lohnliste festhalten kann:

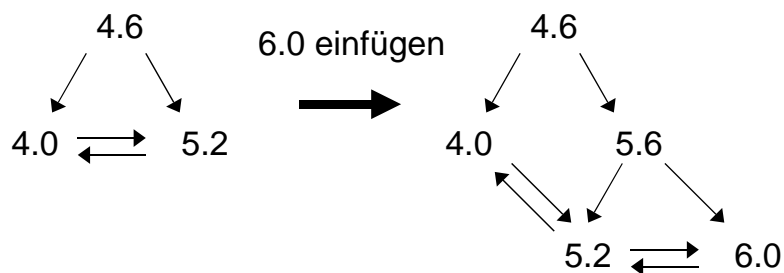
```
salaries.iterate(new Raise(8.8, 1500));
```

Aufgabe 3 (80 min, 16 Punkte)

Spezieller Suchbaum

Der ADT „Menge von float-Werten“ soll imperativ mittels einer Variante des binären Suchbaums repräsentiert werden: Die Werte befinden sich ausschließlich in Blättern; die Nichtblätter enthalten lediglich Werte, die zum „Navigieren“ in Richtung Blätter benutzt werden: wenn ein gesuchter Werte kleiner als der Wert eines betrachteten Knotens ist, wird nach links weitergesucht, andernfalls nach rechts. Zusätzlich sind die Blätter doppelt miteinander verkettet; das ermöglicht z.B. eine schnell Ausgabe der Elemente der Menge in sortierter Reihenfolge. Eingefügt werden könnte beispielsweise so:

Einfügen x: Suche wie in einem normalen Suchbaum. Die Suche endet bei einem Blatt mit Wert y . Falls $x \neq y$, ersetze y durch $(x+y)/2$ und hänge ein linkes Blatt mit $\min(x,y)$ und ein rechtes Blatt mit $\max(x,y)$ an. Passe die doppelte Verkettung der Blätter entsprechend an. Beispiel: Einfügen von 6.0 in die Menge $\{4.0, 5.2\}$:



Gesucht ist die Implementierung dieser Einfügeoperation auf der Grundlage der folgenden Vorgabe:

```
class Floats {
    Node root = null;
    class Node {
        boolean leaf;
        float value;
        Node left, right; // for subtree or linkage
        Node(boolean b, float v, Node l, Node r) {
            leaf = b; value = v; left = l; right = r; }
    }
    public void insert(float x) {.....}
}
```

(Die Angabe von Abstraktionsfunktion, Invariante, Voraussetzung und Effekt ist *nicht* verlangt.)