Informatik I, Blatt Nr. 4, Abgabe: 17.12.2009 um 11 Uhr
http://proglang.informatik.uni-freiburg.de/teaching/info1/2009/

Hinweise zur Abgabe

Bitte reichen Sie Ihre Abgaben bis zum 17.12.2009 um 11 Uhr ein. Abgaben in elektronischer Form schicken Sie **per Email** an **Ihren** Tutor. Abgaben in Papierform werfen Sie bitte in den **Briefkasten** Ihrer Übungsgruppe im Geb. 051 im Erdgeschoss. Bei jeder Aufgabe ist angegeben, ob sie elektronisch oder auf Papier abgegeben werden muss.

Bei allen Aufgaben, die Sie per Mail abgeben, müssen Sie sich an die Namenskonventionen der Aufgaben halten. Dies gilt sowohl für die Dateinamen der Abgabe, als auch für Namen von Funktionen. Bitte geben Sie bei der elektronischen Abgabe nur eine Zip-Datei ab. Diese muss alle in den Aufgaben angegebenen `.scm` Dateien (DrScheme) enthalten. Alle Dateien müssen sich in der Zip-Datei in einem Ordner befinden. Der Name dieses Ordners muss Ihrem Loginnamen für den Rechnerpool des Instituts für Informatik entsprechen. Geben Sie unter keinen Umständen Worddokumente usw. ab!

Achten Sie bei der Papierabgabe darauf, dass jedes Blatt Papier Ihrer Abgabe Ihren Namen, Ihre Übungsgruppe, die Blattnummer und den Namen Ihres Tutors trägt. Falls Ihre Papierabgabe aus mehreren Seiten besteht, tackern Sie die Blätter.

Sie können DrScheme im Pool verwenden (starten mit `drscheme`). Achten Sie darauf, dass Sie jeweils das richtige Sprachlevel ausgewählt haben!

Punktevergabe

Um für die Programmieraufgaben Punkte zu erhalten, folgen Sie den Konstruktionsanleitungen der Vorlesung.

1 Aufgabe

[Sprache: Die Macht der Abstraktion, (1+1+1+1) Punkte]

Aus der Vorlesung ist die Prozedur `filter` bekannt. Benutzen Sie `filter`, um folgende Prozeduren zu implementieren. (Lösungen ohne `filter` oder mit direkter Rekursion werden nicht akzeptiert!)

(a) `(: first-with ((%a -> boolean) (list %a) -> (mixed %a false)))`
`(first-with pred l)` gibt das erste Element der Liste `l` zurück, welches das Prädikat `pred` erfüllt. Gibt es kein solches Element wird `#f` zurückgeliefert.

(b) `(: count-zeroes ((list number) -> natural))`
`(count-zeroes l)` liefert die Anzahl der in `l` enthaltenen Nullen.

(c) `(: any? ((%a -> boolean) (list %a) -> boolean))`
`(any? pred l)` liefert `#t` falls mindestens ein Element von `l` das Prädikat `pred` erfüllt, ansonsten wird `#f` zurückgegeben.

(d) `(: all? ((%a -> boolean) (list %a) -> boolean))`
`(all? pred l)` liefert `#t` falls alle Elemente von `l` das Prädikat `pred` erfüllen, ansonsten wird `#f` zurückgegeben.

Abgabe: elektronisch als `filter.scm`

2 Aufgabe

[Sprache: Die Macht der Abstraktion, (2+2+2) Punkte]

(a) Schreiben Sie eine Prozedur `list-eq?` welche ein Gleichheitsprädikat `p` und zwei Listen `l1` und `l2` nimmt. `(list-eq? p l1 l2)` soll `#t` liefern genau dann wenn `l1` und `l2` gleich sind, wobei die Elemente von `l1` und `l2` mittels `p` verglichen werden.

(b) Implementieren Sie eine endrekursive Variante `filter-it` von `filter`.

(c) Verifizieren Sie experimentell mittels des `check-property` Konstrukts, dass `filter` und `filter-it` immer dasselbe Ergebnis liefern. (Unter der Annahme, dass die übergebenen Listen nicht so groß sind, dass `filter` zu einem Speicherüberlauf führt.)

Abgabe: elektronisch als `filter-it.scm`

3 Aufgabe

[Sprache: Die Macht der Abstraktion, (1+2+1+1) Punkte]

Eine klassische Prozedur höherer Ordnung ist `list-map`, eine Prozedur, die eine Prozedur und eine Liste akzeptiert. Ein Aufruf `(list-map f l)` wendet dann die Prozedur `f` auf jedes Element der Liste `l` an und produziert eine Liste als Rückgabewert. So liefert z.B. ein Aufruf `(list-map f (list x1 x2 x3))` dasselbe Ergebnis wie `(list (f x1) (f x2) (f x3))`.

(a) Definieren Sie zunächst einen möglichst allgemeinen Vertrag für `list-map`.

(b) Implementieren Sie `list-map` (benutzen Sie dazu *nicht* die eingebaute Prozedur `map`).

(c) Implementieren Sie mittels `list-map` eine Prozedur `inc-all`, welche alle Elemente einer Liste von Zahlen um eins erhöht.
Beispiel: `(inc-all (list 1 2 3))` wertet zu `(list 2 3 4)` aus.

(d) Implementieren Sie mittels `list-map` eine Prozedur `check-for-zeroes`, welche alle Elemente einer Liste von Zahlen auf Gleichheit mit Null überprüft.
Beispiel: `(check-for-zeroes (list 0 1 0 2 0 3)` wertet zu `(list #t #f #t #f #t #f)` aus.

Abgabe: elektronisch als `list.scm`

4 Aufgabe

[Sprache: Die Macht der Abstraktion, (2+2+1) Punkte]

(a) Definieren Sie eine Prozedur `tree->list` die einen Baum in eine Liste umwandelt, so dass in der Ergebnisliste für jeden Knoten `(node l b r)` zuerst die Elemente des linken Teilbaums `l`, dann der Wert `b` im Knoten selber und dann die Elemente des rechten Teilbaums `r` auftauchen.
Beispiel:

```
(tree->list (make-node (make-node the-empty-tree 1 the-empty-tree)
                       2
                       (make-node the-empty-tree 3 the-empty-tree)))
```

liefert als Ergebnis `(list 1 2 3)`

(b) Definieren Sie analog zu `list-map` der vorigen Aufgabe eine Prozedur `tree-map`, welche eine Prozedur und einen Binärbaum als Parameter nimmt und die Prozedur auf jedes Element des Binärbaums anwendet.

(c) Benutzen Sie `tree-map` um eine Prozedur `check-for-even`, die einen Binärbaum akzeptiert und alle Stellen um Baum auf Null setzt, an denen eine gerade Zahl steht.

Abgabe: elektronisch als `tree.scm`