Informatik I, Blatt Nr. 9, Abgabe: **19.1.2009** um 11 Uhr
http://proglang.informatik.uni-freiburg.de/teaching/info1/2008/

Hinweise zur Abgabe

Bitte reichen Sie Ihre Abgaben bis zum **19.1.2009** 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. Alle Funktionen müssen einen Vertrag haben (ggf. wie in der Vorlesung als Kommentar) und ausreichend getestet werden!

1 Aufgabe

[Sprache: Die Macht der Abstraktion, 7 Punkte]

Der abstrakte Datentyp `(map1 X)` Speichert Daten nach einem Schlüssel, hier einer Nummer. Mit Hilfe des Schlüssels sollen dann die Daten entsprechend auch wieder aus der `map1` heraus geholt werden können.
Die Verträge für die Operationen von `(map1 X)` sehen wie folgt aus:

- `(: make-empty-map1 (map1 X))`
- `(: map1-empty? ((map1 X) -> boolean))`
- `(: map1-add (natural X (map1 X) -> (map1 X)))`
- `(: map1-find (natural (map1 X) -> X))`

Als Eigenschaften der Operationen muss gelten:

```
(map1-empty? make-empty-map1)  == #t
(map1-empty? (map1-add n y m)) == #f
(map1-find n (map1-add n y m)) == y
(map1-find n (map1-add n y2 (map1-add n y1 m))) == y2
```

(a) Schreiben Sie entsprechend der Vorlesung eine Implementierung für `(map1 X)`. Testen Sie ihre Implementierung ausreichend. Sollte `map1-find` mit einem Schlüssel aufgerufen werden, der nicht vorher in die `map1` eingefügt wurde ist das Verhalten unspezifiziert. Sie dürfen daher solche Aufrufe mit einer `violation` abbrechen.

(b) Erweitern Sie den ADT `(map1 X)` um eine Operation
`(: map1-member (n (map1 X) -> boolean))`
mit folgenden Eigenschaften:

```
(map1-member n make-empty-map1)  == #f
(map1-member n (map1-add n x m)) == #t
```

(c) Erstellen Sie für einen Versandhandel eine `map1` um die Artikelnummern auf die Artikelbezeichnung zu mappen. Fügen Sie dabei folgende Artikel ein:

Artikelnr.	Artikel
123	Pullover rot
316	Teelichter
934	Malblock

Abgabe: Datei `map1.scm`.

2 Aufgabe

[Sprache: Die Macht der Abstraktion, 13 Punkte]

Wir wollen die `map` aus Aufgabe 1 so erweitern, dass sie auch beliebige Schlüssel nehmen kann. Dadurch entsteht ein ADT `(map X Y)` mit zwei Parametern. Die Verträge für die Operationen von `(map X Y)` sind dann:

- `(: make-empty-map ((X X -> boolean) (X X -> boolean) -> (map X Y))`
- `(: map-empty? ((map X Y) -> boolean))`
- `(: map-add (X Y (map X Y) -> (map X Y)))`
- `(: map-find (X (map X Y) -> Y))`
- `(: map-member (X (map X Y) -> boolean))`

Hierbei werden dem Konstruktor wie in der Vorlesung bei `(set X)` die Vergleichsoperatoren `=` und `<` (in dieser Reihenfolge) mit gegeben um die Schlüssel der `map` anordnen zu können.
Als Eigenschaften der Operationen muss gelten:

```
(map-empty? (make-empty-map = <))  == #t
(map-empty? (map-add n y m)) == #f
(map-member n (make-empty-map = <))  == #f
(map-member n (map-add n x m)) == #t
(map-find n (map-add n y m)) == y
(map-find n (map-add n y2 (map-add n y1 m))) == y2
```

(a) Implementieren Sie den ADT `(map X Y)`.

(b) Erstellen sie eine Inventurliste mit Hilfe von `(map X Y)`. Dabei sollen die Schlüssel die Artikelbezeichnungen (Zeichenketten) sein. Fügen Sie folgende Artikel mit Menge ein:

Artikel	Anzahl
Schraube	5126
Holzplatte	12
Klebeband	245
Kompressor	2

Tipp: Verwenden Sie `string<?` und `string=?`.

(c) Überlegen Sie sich weiter sinnvolle Eigenschaften für
`(: map-remove X (map X Y) -> (map X Y))`
und implementieren Sie die Operation.

(d) **Bonus (2 Punkte):** Erweitern Sie den ADT `(map X Y)` um eine Operation
`(: map-list-elems (map X Y) -> (list tupel))`
`map-list-elems` soll alle Elemente der `map` als Liste ausgeben. Definieren Sie sich dafür zuerst einen gemischten Datentyp `tupel` mit den Komponenten `tupel-key` und `tupel-content`.

`map-list-elems` gibt dann eine Liste von `tupel` zurück, wobei die Komponenten entsprechend mit dem Schlüssel und Inhalt der `map` belegt werden.
Geben Sie damit die Inventurliste aus Aufgabe 2b komplett aus.

Abgabe: Datei `map.scm`.