Informatik I, Blatt Nr. 10, Abgabe: 22.1.2009 um 11 Uhr
http://proglang.informatik.uni-freiburg.de/teaching/info1/2008/

Hinweise zur Abgabe

Bitte reichen Sie Ihre Abgaben bis zum 22.1.2009 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, 10 Punkte]

Sei der abstrakte Datentyp `(stack X)` gegeben durch die Verträge der Operationen:

- `(: stack-empty (stack X))`
- `(: stack-empty? ((stack X) -> boolean))`
- `(: stack-push (X (stack X) -> (stack X)))`
- `(: stack-top ((stack X) -> X))`
- `(: stack-pop ((stack X) -> (stack X)))`

Als Eigenschaften der Operationen muss gelten:

```
(stack-empty? stack-empty) == #t
(stack-empty? (stack-push x s)) == #f
(stack-top (stack-push x s)) == x
(stack-pop (stack-push x s)) == s
```

a. Der leere Stack wird durch $\square$ dargestellt, der Stack, in den `5` eingefügt wurde, durch $\boxed{5}$. Zeichnen Sie die Stacks, die sich durch folgende Operationen ergeben, wenn jeweils der vorherige Stack als Parameter übergeben wird:

```
stack-empty, stack-push 4, stack-push 1, stack-push 2, stack-pop,
stack-push 3, stack-pop, stack-pop, stack-push 7, stack-pop, stack-pop
```

b. Schreiben Sie eine Implementierung des Stack-Datentyps. Für die Repräsentation des Stacks soll dabei eine Liste verwendet werden. Implementieren Sie analog zur Vorlesung die Funktionen `list-stack-empty, list-stack-...`. Erstellen Sie dann eine Operationstabelle wie in der Vorlesung vorgestellt (Folien Seite 23 folgende) und stellen Sie den Stack damit, unabhängig Ihrer Implementation, zur Verfügung.

Abgabe: `stack.scm` und auf Papier.

2 Aufgabe

[Sprache: Die Macht der Abstraktion m. Zw., 10 Punkte]

Betrachten Sie folgenden Scheme-Code:

```
(define a1 (make-account 100))
(define a2 (make-account 50))
(define a3 a2)
(account-withdraw a2 60)
(account-withdraw a1 60)
(account-withdraw a3 50)
(account-withdraw a2 50)
(account-balance a1)
(account-balance a2)
(account-balance a3)
```

Werten Sie diesen auf Papier nach dem Substitutionsmodel mit Speicher aus. Orientieren Sie sich an dem Vorgehen in der Vorlesung.

Dabei ist `account` und `account-withdraw` definiert wie in der Vorlesung:

```
; Ein Bankkonto ist ein Wert
; (make-account b)
; wobei b : real der Kontostand ist (veränderlich)
(define-record-procedures-2
  account
  make-account account?
  ((account-balance set-account-balance!)))

; Den Kontostand ändern
(: set-account-balance! (account real -> \%unspecified))
; Effekt: (set-account-balance! a n) setzt den Kontostand auf n

; Geld abheben und anzeigen, ob das möglich war
(: account-withdraw (account real -> boolean))
; Effekt: (account-withdraw a n) ändert den Kontostand von a
(define account-withdraw
  (lambda (a n)
    (if (>= (account-balance a) n)
        (begin
          (set-account-balance! a (- (account-balance a) n))
          #t)
        #f)))
```

Abgabe: Papier.