Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Lehrstuhl Softwaretechnik und Programmiersprachen
Professor Dr. M. Leuschel

21. Februar 2014

Klausur

Funktionale Programmierung
Wintersemester 2013

Nachname: _____ Vorname: _____

Matrikelnummer: _____

Studienfach: _____ Semester: _____

Unterschrift: _____

Zugelassene Hilfsmittel: Keine

Diese Klaujur enthält 9 nummerierte Seiten. Prüfen Sie bitte zuerst, ob alle Seiten vorhanden sind.

Erklärung:

> Hiermit erkläre ich, dass ich mit der Veröffentlichung der Klausurergebnisse unter Angabe von Matrikelnummer und Note bzw. Punktzahl auf der WWW-Seite der Lehrveranstaltung einverstanden bin.

Unterschrift: _____

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	7	8	Σ
Punktzahl	5	10	14	8	12	8	18	6	81
Erreicht									

Aufgabe 1 [5 Punkte]

Kringeln Sie die richtig Antwort ein. Bitte machen Sie klar deutlich, was Ihre Antwort ist. Nicht angekreuzte Fragen werden als falsch bewertet.
Bewertung:

```
(defn punkte [richtig nr_aufgaben]
   (let [p  (- richtig (/ nr_aufgaben 2))]
     (max 0 p)))
```

Tip: Sie können sich durch Raten nicht verschlechtern.

(a) Eine Monade ist
   A. Ein Monoid in der Kategorie der Endofunktoren
   B. Ein Erfrischungsgetränk

(b) Der Y-Kombinator
   A. Findet Fixpunkte von beliebigen Funktionen
   B. Wird benutzt um Rekursion im ungetypen λ Kalkül zu implementieren

(c) `(read-string "`(1)")`
   A. `Attempting to call unbound fn: #'clojure.core/unquote`
   B. `(clojure.core/seq (clojure.core/concat (clojure.core/list 1)))`

(d) Sowohl bind als auch return sind monadische Funktionen
   A. Ja
   B. Nein

(e) In Haskell lässt sich im Gegensatz zu Clojure ein short-circuiting `and` auch ohne Macro ausdrücken. Woran liegt das?
   A. Lazy Evaluation
   B. Monaden

(f) Das Epochal Time Model beschreibt
   A. Den Umgang mit State in Clojure
   B. Einen Weg, die Probleme mit java.util.Date zu beheben

(g) State ist
   A. Der Wert einer Identität zu einem Zeitpunkt
   B. In funktionalen Sprachen generell irrelevant

(h) Wenn eine Sprache als homoikonisch bezeichnet wird, heisst das
   A. sie benutzt eine Präfix Notation
   B. sie wird in ihren eigenen Datenstrukturen notiert

(i) Funktionen höherer Ordnung sind Funktionen,
   A. die mehr als einen Parameter haben
   B. die andere Funktionen als Parameter bekommen oder zurückgeben

(j) Multimethods sind
   A. flexibler als Protokolle
   B. schneller als Protokolle

Aufgabe 2 [10 Punkte]

Folgende Datenstrukturen seien im Namespace `user` gegeben (d.h. Sie haben den Code bereits in der normalen REPL eingegeben)

```
(def xs [1 2 3])
(def q -1)
```

Was ergeben die folgenden Aufrufe, wenn sie in der REPL im Namespace `user` eingegeben werden? Achtung, es könnten auch ungültige Aufrufe dabei sein. Wenn ein Ausdruck einen Fehler produziert, geben Sie an, was die Ursache des Problem ist, Sie brauchen nicht die genaue Fehlermeldung anzugeben.

(a) `` `(+ 2 q) ``

(b) `` `~'q ``

(c) `` `(~@xs ~@xs) ``

(d) `` `(~@xs ~xs) ``

(e) `` `~(`~q) ``

Aufgabe 3 [14 Punkte]

Schreiben Sie eine Funktion, die eine Liste von Tripeln in drei Listen zerlegt.

```
user=> (reorder [[1 2 3] [2 4 5] [6 7 8] [1 2 2]])
[[1 2 6 1] [2 4 7 2] [3 5 8 2]]
```

Aufgabe 4 [8 Punkte]

Gegeben sei folgender Clojure Code

```
(->> (all-ns)
  (map (fn [n] [(str n) (count (ns-interns n))]))
  (sort-by second)
  (map first))
```

Das Programm gibt eine nach Anzahl der enthaltenen Vars sortierte Liste aller Namespaces aus.

Schreiben Sie den Code so um, dass kein Threading Macro benutzt wird.

Aufgabe 5 [12 Punkte]

Schreiben Sie ein Macro, das die logische Implikation berechnet. Beispielsweise berechnet win Aufruf von `(implies a b)` die Implikation $a \implies b$. Der Aufruf `(implies a b c d)` berechnet die Implikation $a \implies (b \implies (c \implies d))$. Sie dürfen **nicht** die anderen Junktoren and, or und not verwenden. Das Macro soll sich so verhalten:

```
user> (implies)
ArityException Wrong number of args (0) passed to: user$implies
user> (implies 1)
1
user> (implies 1 2 3)
3
user> (implies 1 false)
false
user> (implies 1 2 nil)
nil
user> (implies nil 2)
true
```

Aufgabe 6 [8 Punkte]

Wir wollen Mengen in einer symbolischen Darstellung implementieren, d.h., statt einer expliziten Aufzählung der Elemente wollen wir eine Menge als Comprehension $\{x|P(x)\}$ spezifizieren. Der Einfachheit halber interessieren wir uns nur für die `member` Funktion.

(a) [2 Punkte] Definieren Sie ein Protokoll PSet, das ein Prädikat `member?` definiert. Die Verwendung von `member?` sehen Sie in b) und c)

(b) [3 Punkte] Implementieren Sie ein Record SetComprehension, das

- Ein Prädikat P als Parameter bekommt
- Das PSet Protokoll implementiert
- Die Menge $\{x \mid P(x)\}$ repräsentiert

```
user=> (def s (->SetComprehension even?))
#'user/s
user=> (member? e 3)
false
user=> (member? e 4)
true
```

(c) [3 Punkte] Implementieren Sie PSet für das Standard Clojure Set[1].

```
user=> (def t #{1 2 3})
#'user/t
user=> (member? t 1)
1
user=> (member? t 4)
nil
```

[1]Fragen Sie nicht welche Klasse das ist. Entweder Sie wissen es auswendig oder Sie finden einen anderen Weg das Problem zu lösen.

Aufgabe 7 [18 Punkte]

Natürliches Mergesort ist eine Mergesort Variante, die bereits sortierte Subsequenzen ausnutzt. Schreiben Sie eine Funktion, die eine Sequenz von Zahlen bekommt und eine Sequenz der Teilsequenzen liefert. Ihre Funktion muss eine polynomiale Laufzeit haben

Zur Erleichterung können Sie annehmen, dass dieselbe Zahl niemals mehrfach direkt hintereinander auftritt und die Eingabsequenz niemals leer ist.

Beispielaufruf:

```
(runs [1 4 6 2 8 10 3 1 7])
=> ((1 4 6) (2 8 10) (3) (1 7))
```

Aufgabe 8 [6 Punkte]

Bonusaufgabe

Gegeben sei folgender Code, der eine Funktion `xyx` definiert.

```
(def m sequence-m)

(defn xyz [a b]
  (with-monad m
     ((m-lift 1 a) b)))
```

(a) [3 Punkte] Eigentlich gibt es in Clojure schon eine Funktion, die das Gleiche wie `xyz` berechnen kann. Um welche Funktion handelt es sich?

(b) [3 Punkte] Geben Sie sowohl einen Beispielaufruf für `xyz` als auch den äquivalenten Clojure Aufruf an. Was ist das Ergebnis des Aufrufs?

Ihr Beispielaufruf darf einfach, aber nicht trivial sein. Trivial wäre es zum Beispiel bei einer rekursiven Funktion den Basisfall aufzurufen, bei einer Higher Order Funktion die Identitätsfunktion zu übergeben oder bei einer Funktion, die Listen verarbeitet die leere Liste zu benutzen.