Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Lehrstuhl Softwaretechnik und Programmiersprachen
Dr. Philipp Körner

08. Februar 2024

Klausur

Einführung in die Funktionale Programmierung
Wintersemester 2023/24

Nachname: _____ Vorname: _____

Matrikelnummer: _____

Zugelassene Hilfsmittel: verkürztes Clojure Cheat Sheet
Dauer: 90 Minuten

Diese Klaujur enthält 7 nummerierte Seiten. Prüfen Sie bitte zuerst, ob alle Seiten vorhanden sind. Mit Ihrer Unterschrift versichern Sie, eine vollständige Klausur erhalten zu haben.

Unterschrift: _____

Schalten Sie bitte jegliche elektronischen Geräte aus

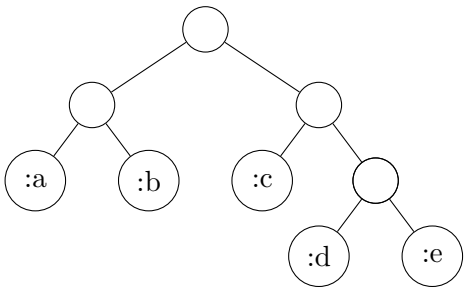
Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	Σ
Punktzahl	17	13	20	15	20	15	100
Erreicht							

Aufgabe 1 [17 Punkte]

(a) [11 Punkte] Ein Binärbaum, der Keywords speichert, kann als Map dargestellt werden, bei dem man unter dem Schlüssel `:left` den linken Teilbaum und unter `:right` den rechten Teilbaum speichert. Blätter sind direkt Keywords. Im folgenden Beispiel stellt die Map links den Binärbaum rechts dar.

```
{:left  {:left :a
         :right :b}
 :right {:left :c,
         :right {:left :d,
                 :right :e}}}
```



Schreiben Sie eine Funktion `(iso? tree1 tree2)` die feststellt, ob die beiden Bäume `tree1` und `tree2` den gleichen Aufbau an Knoten haben und sich höchstens in den Werten ihrer Blätter unterscheiden.

Beispielaufrufe:

```
user=> (iso? {:left :foo, :right {:left :bar, :right :quux}}
             {:left :a,   :right {:left :b,   :right :c}})
true ;; es existiert die Abbildung foo -> a, bar -> b, quux -> c

user=> (iso? {:left :a,  :right {:left :b, :right :c}}
             {:left {:left :b, :right :c}, :right :a})
false ;; tree1 hat links ein Blatt, tree2 einen inneren Knoten

user=> (iso? :a :b)
true ;; auf oberster Ebene sind beide nur ein Blatt
```

Hinweis: Achten Sie darauf, dass Ihre Implementierung in allen Fällen terminiert.

(b) [6 Punkte] Gegeben sei die folgende Implementierung, die das Produkt einer Liste von Zahlen berechnet:

```
(defn mult [l]
  (if (empty? l)
    1
    (*' (first l)
        (mult (rest l)))))
```

Welches Problem hat diese Implementierung?[3P] Geben Sie eine andere Version an, die dieses Problem löst.[3P]

Aufgabe 2 [13 Punkte]

In dieser Aufgabe soll ein Macro zur Abfrage aus einer Datenstruktur implementiert werden. Die Form eines Aufrufs soll wie folgt sein:

`(select identifier :from data :if pred)`

Die Argumente an `select` sind dabei:

- `identifier` ist ein beliebiges Symbol.
- `:from` und `:if` ist Syntax und wird immer als exaktes Keyword übergeben. Dies muss nicht verifiziert werden.
- `data` enthält eine Seqable Datenstruktur.
- `pred` ist ein Ausdruck, der von `identifer` abhängen darf.

Rückgabe soll eine Teilsequenz (in Reihenfolge) sein, die genau die Elemente enthält, die das Prädikat erfüllen.

Beispielaufrufe:

```
user=> (select item :from [10 20 25 15 30 12 23 5] :if (>= item 20))
(20 25 30 23)
user=> (def v [1 2 3])
user=> (select x :from v :if (even? x))
(2)
```

(a) [4 Punkte] Warum kann man `select` nicht als Funktion schreiben?[2P] Begründen Sie Ihre Antwort.[2P]

(b) [3 Punkte] Geben Sie an, wogegen der Aufruf `(select x :from [1 2 3] :if (even? x))` expandieren soll.

(c) [6 Punkte] Implementieren Sie `select`.

Aufgabe 3 [20 Punkte]

(a) [3 Punkte] Was bedeutet es, wenn eine Implementierung des epochalen Zeit-Modells *koordiniert* ist?[2P] Welche reference types sind koordiniert?[1P]

(b) [3 Punkte] Was bedeutet es, wenn eine Implementierung des epochalen Zeit-Modells *asynchron* ist?[2P] Welche reference types sind asynchron?[1P]

(c) [5 Punkte] Bei welchem reference type kann es zum write skew kommen?[1P] Wie viele Objekte des Typen benötigt man dafür mindestens?[2P] Begründen Sie Ihre Antwort.[2P]

(d) [6 Punkte] Gegeben sei der folgende Code:

```
(def foo (ref 100))
(defn bar [x]
  (dosync
    (if (<= x @foo)
      (alter foo - x)
      @foo)))
```

1. Verwendet die Operation innerhalb des `if` die Ref `foo` konsistent?[1P] Falls nein, geben Sie eine korrigierte Fassung an. Begründen Sie *kurz* Ihre Antwort.[2P]
2. Kann ein write skew auftreten?[1P] Falls ja, geben Sie eine korrigierte Fassung an. Begründen Sie *kurz* Ihre Antwort.[2P]

(e) [3 Punkte] Gegeben sei der folgende Code:

```
(def foo (atom {:value 100}))
(defn bar [logging?]
  (if logging?
    (println @foo)
    (swap! foo assoc :log logging?)))
```

Verwendet die Operation innerhalb des `if` das Atom `foo` konsistent?[1P] Falls nein, geben Sie eine korrigierte Fassung an. Begründen Sie *kurz* Ihre Antwort.[2P]

Aufgabe 4 [15 Punkte]

Es soll eine sequenzielle Datenstruktur entwickelt werden, die endlich viele Elemente speichert. Dazu merkt man sich die Länge und eine Basisdatenstruktur, wie etwa eine Clojure Liste. Diese Liste ist *mindestens* so lang, wie die Sequenz, die repräsentiert wird; alle zusätzlichen Elemente werden ignoriert. Beispiele, wie die Sequenz `(1 2 3)` repräsentiert werden kann, sind also:

`{:length 3, :list (1 2 3)}` und `{:length 3, :list (1 2 3 :ignored :who-cares)`.

Eine Implementierung von `(add e data)`, die in konstanter Laufzeit ein Element vorne anfügt, kann also wie folgt aussehen:

```
(defn add [e data]
  {:length (inc (:length data))
   :list (cons e (:list data))})
```

(a) [4 Punkte] Schreiben Sie eine Funktion `(eq data1 data2)`, die testet, ob die gleichen Sequenzen dargestellt werden.

(b) [3 Punkte] Schreiben Sie eine Funktion `(fst data)`, die in konstanter Laufzeit das erste Element zurückgibt. Ist die Länge 0, so soll `nil` zurückgegeben werden.

(c) [3 Punkte] Schreiben Sie eine Funktion `(droplast data)`, die in konstanter Laufzeit das letzte Element entfernt.

(d) [5 Punkte] Fertigen Sie eine Skizze an, die das Teilen von Struktur beim Aufruf `(add :x {:length 2, :list '(1 2 3)}` hervorhebt. Es müssen die gesamten Datenstrukturen aus der Skizze hervorgehen.

Aufgabe 5 [20 Punkte]

(a) [6 Punkte] In der Objekt-orientierten Programmierung "complecten" Objekte mehrere Komponenten des epochalen Zeit-Modells. Welche Komponenten sind das?[1P] Geben Sie jeweils das Äquivalent der Komponenten in der OOP an.[5P]

(b) [8 Punkte] Angenommen, Sie sollen einen Schachcomputer implementieren. Welche der folgenden Aspekte sind essentielle Komplexität? Welche sind nicht-essentiell? Begründen Sie jeweils kurz Ihre Antwort.[je 2P]

1. Speicherverwaltung via `malloc` und `free`.
2. Das Bewegungsmuster des Springers.
3. Graphische Ausgabe via JavaFX.
4. Spezialisierte Implementierung von bekannten Eröffnungen.

(c) [6 Punkte] Geben Sie je ein Beispiel für etwas, was *simple* und *easy* ist. Die Beispiele müssen nicht auf Code bezogen sein und dürfen aus der echten Welt stammen. Begründen Sie kurz die Korrektheit Ihrer Beispiele.

Aufgabe 6 [15 Punkte]

Wir wählen als Darstellung für eine Menge Reis eine Map mit den Schlüsseln `:art` und `:menge` (in Gramm). Beispielsweise würde man 100 Gramm Basmati-Reis darstellen als: `{:art :basmati, :menge 100}`

Die Funktion `cook` soll die Anweisungen zum Reiskochen zurückgeben. Diese enthalten zusätzlich die Schlüssel `:flsg` (Flüssigkeit) und `:f-menge` (Flüssigkeitsmenge in ml).

(a) [9 Punkte] Implementieren Sie `cook` für folgende Reissorten. Die Funktion *muss* ohne Re-Definition um zusätzliche Sorten erweiterbar sein.[5P]

- Der Typ `:basmati` benötigt 1.5-mal so viel ml Wasser wie die Gramm-Menge des Reis.[2P]
- Der Typ `:milchreis` benötigt 5-mal so viel ml Milch wie die Gramm-Menge des Reis.[2P]

Beispielaufrufe:

```
user=> (cook {:art :basmati, :menge 100})
{:art :basmati, :menge 100, :flsg :wasser, :f-menge 150}
user=> (cook {:art :milchreis, :menge 200})
{:art :milchreis, :menge 200, :flsg :milch, :f-menge 1000}

user=> ??? ;; Erweiterung durch den Nutzer
user=> (cook {:art :risotto, :menge 80})
{:art :risotto, :menge 80, :flsg :brühe, :f-menge 300}
```

(b) [6 Punkte] Skizzieren Sie, wie man `cook` (ggf. mit Anpassungen beim Aufruf) als Protokoll implementieren könnte. Sie müssen keinen vollständigen Code angeben. Es muss aber klar werden, wie die Signatur aussieht[2P] und wie man verschiedene Reissorten[2P] und konkrete Reismengen darstellt[2P].