Grundlagen der Informatik 1

Sommersemester 2011

Dr. Guido Rößling
https://moodle.informatik.tu-darmstadt.de/

Übung 3 Version: 1.0 02. 05. 2011

---

1 Mini Quiz

Kreuzen Sie die wahren Aussagen an!

☐ Strukturdefinitionen können mit **local** innerhalb von Funktionen erfolgen.

☐ Eine Baumstruktur kann in Racket verschiedene Datentypen in ihren Knoten speichern.

☐ Innerhalb eines **local** Ausdrucks kann nicht auf Definitionen außerhalb des **local** Ausdruck zugegriffen werden.

☐ Der Scope einer Namensbindung ist der textuelle Bereich, in dem sich ein Auftreten des Namens auf diese Namensbindung bezieht.

2 Fragen

1. Welche Richtlinien sollten bei der Benutzung von **local** beachtet werden?
2. Was macht aus Software Engineering-Sicht hochwertigen Code aus?
3. Erläutern Sie Probleme, die entstehen können, wenn Daten redundant abgelegt werden.

3 Nutzung von local

Der RGB-Code eines Farbtons ist ein Tripel (R, G, B) mit $R, G, B \in [0; 255]$. Die Werte R, G und B sind dabei die Anteile der Grundfarben Rot, Grün und Blau am Farbton. Sie reichen von 0 (nicht im Farbton vorhanden) bis 255 (in voller Intensität im Farbton vorhanden). So steht (255 0 0) für reines Rot, (114 247 160) für Hellgrün.
Der RGB-Code eignet sich gut, um Farben für die Darstellung auf einem Bildschirm zu speichern. Der vom Menschen empfundene Farbton hingegen lässt sich aus einem RGB-Tripel nur schwer ersehen. Was für eine Farbe ist z.B. (204 102 153)?
Um dieses Problem zu lösen, kann man den Winkel H (H für hue, Farbton) zu einem RGB-Tripel berechnen. Ein H-Winkel in der Nähe von 0 bedeutet z.B. rötlich, in der Nähe von 240 bedeutet bläulich. Aus Wikipedia stammen folgende Formeln zur Errechnung des Winkels H zu einem RGB Tripel (R, G, B):

$$
\begin{aligned}
r &= R/255 \\
g &= G/255 \\
b &= B/255 \\
M &= \max(r,g,b) \\
m &= \min(r,g,b)
\end{aligned}
$$

$$
H = \begin{cases} 0^\circ & \text{für } M = m, \\ 60^\circ \times \frac{g-b}{M-m} & \text{für } r = M, \\ 120^\circ + 60^\circ \times \frac{b-r}{M-m} & \text{für } g = M, \\ 240^\circ + 60^\circ \times \frac{r-g}{M-m} & \text{für } b = M \end{cases}
$$

Falls der nach der Formel berechnete H-Wert negativ ist, ist 360 zu ihm zu addieren.
Racket nutzt eine Struktur color mit den Feldern *red, blue* und *green* zum Speichern von RGB-Tripeln. Anbei finden Sie die Lösung einer Implementierung der *hue*-Funktion, die aus einer als color-Struktur übergebenen RBG-Farbe den H-Wert berechnet.

```
;; Diese Funktion wandelt eine RBG-Farbe in HSL um
;; http://de.wikipedia.org/wiki/HSI-Farbmodell
;; http://de.wikipedia.org/wiki/RGB-Farbraum
;; hue: color -> number
;; converts the RBG color coded into the hue
;; example: (hue (make-color 0 0 0)) is 0
(define (hue color)
  (cond
    [(= (max (/ (color-red color) 255) (/ (color-green color) 255)
             (/ (color-blue color) 255))
        (min (/ (color-red color) 255) (/ (color-green color) 255)
             (/ (color-blue color) 255)))
     0]
    [(= (/ (color-red color) 255)
        (max (/ (color-red color) 255)
             (/ (color-green color) 255)
             (/ (color-blue color) 255)))
     (* 60(/ (- (/ (color-green color) 255) (/ (color-blue color) 255))
             (- (max (/ (color-red color) 255)
                     (/ (color-green color) 255)
                     (/ (color-blue color) 255))
                (min (/ (color-red color) 255)
                     (/ (color-green color) 255)
                     (/ (color-blue color) 255)))))]
    [(= (/ (color-green color) 255)
        (max (/ (color-red color) 255)
             (/ (color-green color) 255)
             (/ (color-blue color) 255)))
     (+ 120 (* 60 (/ (- (/ (color-blue color) 255)
                        (/ (color-red color) 255))
             (- (max (/ (color-red color) 255)
                     (/ (color-green color) 255)
                     (/ (color-blue color) 255
                        ))
                (min (/ (color-red color) 255)
                     (/ (color-green color) 255)
                     (/ (color-blue color) 255)))))))]
    [(= (/ (color-blue color) 255)
        (max (/ (color-red color) 255)
             (/ (color-green color) 255)
             (/ (color-blue color) 255)))
     (+ 240 (* 60 (/ (- (/ (color-red color) 255)
                        (/ (color-green color) 255))
             (- (max (/ (color-red color) 255)
                     (/ (color-green color) 255)
                     (/ (color-blue color) 255))
                (min (/ (color-red color) 255)
```

```
48                    (/ (color-green color)  255)
49                    (/ (color-blue color)  255))))))]))
```

Ihre Aufgabe ist es nun, aus dieser Lösung eine *sinnvolle* Lösung zu machen, indem Sie über **local** gleichlautende Codeteile—Prozeduren oder einmalig berechenbare Werte—*sinnvoll* in **local**-Blöcken berechnen.
Hinweis: Nutzen Sie einen Bleistift und ein Blatt Papier, um zunächst eine Liste der „mehrfach berechneten aber identischen Werte“ zu bestimmen. Anhand dieser Liste können Sie dann einfach bestimmen, welche Elemente in einem (oder in mehreren geschachtelten) **local**-Blöcken definiert werden sollten.
Hinweis: Wenn Sie den Code testen wollen, sollten Sie in DrRacket über das Menü *Sprache* bzw. *Language* das TeachPack „image.ss“ hinzufügen.

4 Tree Sort

In dieser Aufgabe werden Sie den TreeSort-Algorithmus zum Sortieren von Elementen implementieren. Dazu wird eine in der Informatik sehr oft verwendete Datenstruktur benötigt: der Binärbaum.

Hintergrund: Die Datenstruktur sortierter Binärbaum

Ein Binärbaum ist eine rekursive Datenstruktur, die folgendermaßen definiert ist: Jeder *Baumknoten* enthält einen Wert, einen linken Sohn und einen rechten Sohn. Der linke und rechte Sohn sind dabei auch wieder *Baumknoten*. Als Rekursionsanker dient der „leere Baum“, **empty**, der per Definition ein Baumknoten ist. Sind beide Söhne eines Baumknotens **empty**, so nennt man diesen Baumknoten ein „Blatt“. Der oberste Baumknoten, der selber nicht Sohn eines weiteren Baumknotens ist, heißt „Wurzel“ des Baumes.
Zusätzlich gilt bei einem sortierten binären Baum folgende Bedingung: Sei n ein Baumknoten. Der *linke Sohn* von n ist entweder ein leerer Baum (**empty**) oder ein sortierter Binärbaum, dessen größtes Element kleiner oder gleich dem Wert von n ist. Der *rechte Sohn* ist entweder ein leerer Baum (**empty**) oder ein sortierter Binärbaum, dessen kleinstes Element größer als der Wert von n ist. Vergleichen Sie dazu folgende Racket-Definition[1] und die Beispiele:

```
1  ;; struct for storing binary trees
2  (define-struct tree-node (left content right))
3
4  ;;example binary tree with three nodes
5  ;;  2
6  ;; /  \
7  ;; 1  3
8  ;; /\ /\
9  ;;
10 (make-tree-node (make-tree-node empty 1 empty) 2
11                 (make-tree-node empty 3 empty))
12
13 ;;not a sorted binary tree! Left son is larger than the node content!
14 ;;  2
15 ;; /  \
16 ;; 4  3
17 ;; /\ /\
18 ;;
19 (make-tree-node (make-tree-node empty 4 empty) 2
20                 (make-tree-node empty 3 empty))
```

[1] in der Vorlage sind die Werte nicht „verrutscht“; das kommt nur durch die Umsetzung für das Übungsblatt!

Sortieren von Zahlen

Lesen Sie zunächst aufmerksam alle Teilaufgaben durch, bevor sie mit der Implementierung der Lösung beginnen. Entscheiden Sie sich für ein Vorgehen—top-down oder bottom-up, siehe T3.30—und machen Sie sich eine Wunschliste von Funktionen. Verwenden Sie **local** für Funktionen, die nicht nach außen sichtbar sein sollen.

1. (K) Implementieren Sie eine Funktion tree−insert, die die Wurzel $root$ eines sortierten Binärbaums T und eine Zahl n übergeben bekommt und die die Wurzel eines neuen sortierten Binärbaums T' liefert, der alle Elemente von T sowie n enthält. Beachten Sie dabei die Regeln für sortierte binäre Bäume: der Wert eines linken Sohns muss immer *kleiner oder gleich* dem Wert des Vaters sein, und der Wert eines rechten Sohns muss immer *größer* als der Wert des Vaters sein!

2. Implementieren Sie eine Funktion tree−insert−**list**, die eine Liste von Zahlen in einen sortierten Binärbaum einfügt.

3. Implementieren Sie eine neue Funktion sort−**list**, die alle Zahlen in einer Liste zuerst in einen sortierten Binärbaum einfügt und dann als sortierte Liste wieder zurückgibt.

4. Stellen Sie Ihren Code nun so um, dass lediglich die Prozedur sort−**list** von Außen sichtbar ist, alle Hilfsfunktionen hingegen lokal sind.

5. Angenommen, wir haben folgende Strukturdefinition:

   (**define**−**struct** student (name id birth−year birth−month birthday))

   Was muss im Code geändert werden, um Studierende nach der *ID* bzw.—in einer separaten Funktion—nach dem *Geburtsjahr* zu sortieren?

Hausübung

Die Vorlagen für die Bearbeitung werden im Lernportal Informatik bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im Lernportal Informatik abgegeben werden.
Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren.
Falls Sie die Hausübung in einer Lerngruppe bearbeitet haben, geben Sie dies bitte deutlich bei der Abgabe an. Alle anderen Mitglieder der Lerngruppe müssen als Abgabe einen Verweis auf die gemeinsame Bearbeitung einreichen, damit die Abgabe im Lernportal Informatik auch für sie bewertet werden kann. Beachten Sie dazu die Hinweise bei der Aufgabenabgabe im Lernportal Informatik!

Abgabedatum: Freitag, 13. 05. 2011, 16:00 Uhr

Denken Sie bitte daran, Ihren Code hinreichend gemäß den Vorgaben zu kommentieren (Racket: Vertrag, Beschreibung und Beispiel sowie zwei Testfälle pro Funktion; Vertrag, Beschreibung und Beispiel für jede **local** definierte Funktion; Vertrag (ohne Namen) und kurze Beschreibung für jeden **lambda**-Ausdruck; Java: JavaDoc). Zerlegen Sie Ihren Code sinnvoll und versuchen Sie, wo es möglich ist, bestehende Funktionen wiederzuverwenden. Wählen Sie sinnvolle Namen für Hilfsfunktionen und Parameter.
Verwenden Sie als Sprachlevel für die gesamte Hausübung „Zwischenstufe".

5 Sortieren von Emails (7 Punkte)

Um die Email-Flut zu bewältigen, sortieren viele Nutzer ihre Emails nach verschiedenen Kriterien vor. Die meisten ernstzunehmenden Mailprogramme bieten hierzu mehr oder weniger ausgefeilte Regelwerke an, mit denen dies automatisch geschehen kann. Eine besonders beliebte Vorfilterung ist die Klassifikation von Emails als „Spam" durch externe Programme wie *Apache SpamAssassin*[2].

[2] `http://spamassassin.apache.org/`

In dieser Aufgabe werden wir Emails nach den folgenden Kriterien in lediglich vier verschiedene Ordner vorsortieren. Bitte lesen Sie die Teilaufgaben *genau*, damit Sie nicht sinnlos Punkte verschenken!

1. Emails die im Betreff als [SPAM] oder [SPAM?] klassifiziert werden, werden in den Ordner *Spam* (spam) verschoben.
2. Emails mit einem nicht im Adressbuch verzeichneten Absender werden in einen Ordner *Unbekannt* (unknown) verschoben.
3. Emails die den Nutzer indirekt—das heißt nicht über eine seiner verzeichneten Adressen in der „`An:`"-Zeile—erreicht haben, wandern in den Ordner *Indirekt* (indirect).
4. Alle anderen Emails sind wohl kein Spam (siehe 1.) und kommen von einem Bekannten an eine bekannte Adresse. Diese kommen daher in den Ordner *Freunde* (friends).

Im Folgenden werden wir dies schrittweise implementieren.
Die Aufgaben verwenden dabei die Strukturen email für Emails mit Absender- und Zieladresse, Betreff und Inhalt, contact für Kontakte (mit Name und Email-Adresse) sowie mail−folders mit den vier Listen *friends, indirect, unknown* und *spam* für die vier Verzeichnisse. Die Strukturen sind wie folgt definiert:

```
1  ;; email represents an email
2  ;; from: String − email address of sender
3  ;; to: String − email address of recipient
4  ;; subject: String − subject line
5  ;; contents: String − the contents of the email
6  (define−struct email (from to subject contents))
7
8  ;; contact represents a single contact
9  ;; name: String − the name of the contact
10 ;; address: String − the email address
11 (define−struct contact (name address))
12
13 ;; mail−folders represents the folders for an email program
14 ;; friends: (listof email) − emails from friends
15 ;; indirect: (listof email) − emails received but not sent via "To:"
16 ;; unknown: (listof email) − emails from persons not in the contacts
17 ;; spam: (listof email) − emails classified as spam or possible spam
18 (define−struct mail−folders (friends indirect unknown spam))
19
20 ;; example data
21 (define karl (make−contact "Karl Klammer" "karl@klammer.com"))
22 (define eva (make−contact "Eva Longoria" "eva@longoria.com"))
23 (define my−contacts (list karl eva))
24
25 (define my−emails (list "me@me.com" "me@acm.org" "me@gmx.net"))
```

5.1 Beispiele zur Erläuterung (ohne Punkte)

Im obigen Code werden zwei bekannte Absender—mit den Emailadressen `karl@klammer.com` und `eva@longoria.com`—sowie die bekannten eigenen Emailadressen—`me@me.com, me@acm.org` sowie `me@gmx.net`—definiert.
Das Verhalten Ihres Programms sollte dann wie folgt sein. *Achtung:* diese Beispiele zählen *nicht* zu den „selbst zu schreibenden Tests", sollten von Ihnen aber als Basistests genutzt werden!

1. Jede Email mit einem Betreff, der mit [SPAM] oder mit [SPAM?] beginnt, wird unabhängig von der Absender- und Empfangsaddresse in jedem Fall in den Ordner spam einsortiert.
2. Eine Email, die von einem unbekannten Absender—etwa von k@arl.de—stammt aber nicht in den Ordner spam einsortiert wurde, wandert in den Ordner unknown.

3. Eine Email mit bekanntem Absender, die kein Spam ist aber deren Zieladresse nicht zu den bekannten Empfangsaddressen zählt—also etwa an `me@me.de` gerichtet war—wandert in den Ordner indirect.

4. Schließlich wird jede bislang noch nicht wegsortierte Email in den Ordner friends einsortiert, da sie kein Spam ist (1. Kriterium), von einem bekannten Absender stammt (2. Kriterium) und an eine gültige eigene Adresse gerichtet war (3. Kriterium).

5.2 Prüfung auf bekannte Absenderadresse (1 Punkt)

In dieser Teilaufgabe ist die Absenderadresse darauf zu prüfen, ob sie identisch zu einer Adresse im Adressbuch ist. Schreiben Sie hierzu eine Prozedur known−sender?. Die Prozedur konsumiert einen **String**—die Zieladresse einer Email—und eine Liste von contact-Strukturen—die bekannten Email-Adressen in den Kontakten. Das Ergebnis soll genau dann **true** sein, wenn der übergebene String in den Adressen der Liste im address-Feld enthalten ist.

5.3 Prüfung auf gültigen Empfänger (1 Punkt)

Schreiben Sie eine Prozedur valid−recipient?. Diese konsumiert eine Email-Adresse (als **String**) sowie eine Liste der gültigen Email-Adressen des Empfängers als (listof **String**), etwa my−addresses aus der Vorlage. Die Prozedur soll genau dann **true** liefern, wenn die Empfängeradresse in der Liste der Adressen enthalten ist.

5.4 Prüfung der Email auf Spam (2 Punkte)

Eingehende Emails werden durch das Hilfsprogramm *SpamAssassin* automatisch als Spam oder möglicher Spam markiert. Die Markierung erfolgt, indem dem normalen Betreff einer Email der Text [SPAM] bei eindeutigem Spam bzw. bei Spamverdacht [SPAM?] vorangestellt wird.
Schreiben Sie dazu eine Prozedur is−spam?, die einen String konsumiert und **true** genau dann liefert, wenn der Betreff (subject) der Email mit [SPAM?] oder [SPAM] beginnt.
Hinweis: Nutzen Sie die Prozeduren substring: **String number number** −> **String**, um einen Teilstring zu erzeugen. Die Zählung beginnt bei 0; der Teilstring enthält alle Zeichen von der ersten bis zur zweiten Position *einschließlich*. So liefert (substring "Hello" 2 5) das Ergebnis `"llo"`.
Hinweis: Prüfen Sie *vor* der Nutzung von substring die Länge des Betreffs, um einen Fehler durch ungültige Bereichsangaben zu vermeiden. Verwenden Sie dazu **string−length**: **String** −> **number**. So liefert (**string−length** "Hello") 5.

5.5 Sortierung der Emails (3 Punkte)

Schreiben Sie eine Prozedur sort−emails: (listof email)mail−folders (listof contact)(listof **String**)−> mail−folders. Diese konsumiert die Liste der neu eingetroffenen Mails, den aktuellen Zustand der Email-Ordner, die Liste der bekannten Kontakte (für die Prüfung der Absenderadresse) sowie die Liste der eigenen gültigen Empfangsadresse (für die Prüfung der Empfängeradresse). Die Prozedur liefert den neuen Zustand der Email-Ordner.
Implementieren Sie die Prozedur gemäß den Vorgaben am Anfang der Aufgabe! Die Reihenfolge der Prüfschritte ist *strikt einzuhalten*, um die Emails korrekt zu sortieren. Andernfalls käme beispielsweise eine Werbe-Email von der Adresse eines Freundes in den Ordner friends statt in spam.
Hinweis: Hier kann es sehr sinnvoll sein, mit einer oder mehreren lokalen Hilfsprozedur(en) zu arbeiten, um nicht für jede „einsortierte“ Email eine neue mail−folders-Instanz erstellen zu müssen.
Zu beachten: Die Emails sind der Reihenfolge der Abarbeitung in die—nicht zwangsweise leeren—Ordner einzufügen. Sie sollen *auf keinen Fall* umsortiert werden, so dass am Ende etwa die älteste eingetroffene Email vorne steht!