Grundlagen der Informatik 1

Wintersemester 2009/2010

Prof. Dr. Max Mühlhäuser, Dr. Rößling
http://proffs.tk.informatik.tu-darmstadt.de/teaching

Übung 2 Version: 1.0 26. 10. 2009

---

1 Mini Quiz

☐ Elemente einer Liste können von verschiedenen Datentypen sein.

☐ Rekursive Datentypen und Prozeduren brauchen zum Terminieren immer einen Rekursionsanker.

☐ Man kann mit `cons` Listen erzeugen, die man mit `list` nicht erzeugen kann.

☐ Strukturdefinitionen erzeugen automatisch Konstruktoren, Selektoren und Prädikate.

☐ Natürliche Zahlen sind abgeschlossen unter der Subtraktion.

2 Fragen

1. Beschreiben Sie mit eigenen Worten, was ein rekursiver Datentyp ist.
2. Erklären Sie, wieso rekursive Datenstrukturen in der Praxis nicht unendlich groß werden.
3. Welche zwei Ansätze beim Design von Programmen mit Hilfe von Wunschlisten kennen Sie? Diskutieren Sie Vor- und Nachteile des jeweiligen Ansatzes.
4. Erklären Sie mit eigenen Worten, was Abgeschlossenheit ist.

3 Strukturen (K)

Versuchen Sie bitte, die Aufgaben erst ohne einen Rechner zu lösen.
Gegeben sei folgende Strukturdefinition:

```
(define-struct my-pair (first second))
```

Werten Sie folgende Ausdrücke aus und geben Sie das Ergebnis an.

1. `(make-my-pair 'a 'b)`
2. `(my-pair? (make-my-pair 'a 'b))`
3. `(my-pair? (list 'a 'b))`
4. `(make-my-pair 1 (make-my-pair 2 empty))`
5. `(* (my-pair-second (make-my-pair 1 2)) (my-pair-first (make-my-pair 3 4)))`

4 Listen (K)

Nehmen Sie für die folgenden Aufgaben das Sprachlevel „Anfänger mit Listen-Abkürzungen“ an. Versuchen Sie bitte, die Aufgaben erst ohne einen Rechner zu lösen. Diese Aufgabe sollte bearbeitet werden, um die Hausübung lösen zu können.

1. Werden die folgenden Ausdruckspaare zu äquivalenten Listen ausgewertet? Was ergibt die Auswertung jeweils?
   a) (**cons** 1 (**cons** 2 (**cons** 3 **empty**))) und (**list** 1 2 3 **empty**)
   b) (**cons** (**list** '()) **empty**) und (**list** ' **list** **empty**)
   c) (**list** 7 '∗ 6 '= 42) und (**cons** 7 (**cons** '∗ (**cons** 6 (**cons** '= (**list** 42)))))
   d) (**cons** 'A (**list** '(I))) und (**list** 'A (**cons** 'I **empty**))
2. Werden die folgenden Ausdrücke jeweils fehlerfrei ausgewertet? Falls nicht, begründen Sie bitte, was zum Fehler führt.
   a) (**cons** 1 (**cons** 2 (**cons** 3)))
   b) (**cons** 1 (**list** 2 (**list** '(3 + 4))))
   c) (**list** (**cons empty** 1)(**cons** 2 **empty**)(**cons** 3 **empty**))
3. In der Vorlesung haben Sie gesehen, wie man Listen in Kästchenschreibweise darstellt (siehe T3.8). Stellen Sie folgende Listen in Kästchenschreibweise dar:
   a) (**define** A (**list** (**cons** 1 **empty**)(**list** 7) 9))
   b) (**define** B (**cons** 42 (**list** 'Hello 'world '!)))
4. Zu welchen Werten werden die folgenden Ausdrücke (mit A und B wie oben definiert) ausgewertet? Falls ein Auswertung nicht möglich ist, begründen Sie das bitte.
   a) (**first** (**rest** A))
   b) (**rest** (**first** A))
   c) (**first** **empty**)
   d) (**append** (**first** B) (**rest** (**rest** A)) (**first** A))
5. Im folgenden sollen Sie rekursive Prozeduren auf Listen definieren. Vergessen Sie nicht, zuerst Vertrag, Beschreibung und ein Beispiel anzugeben.
   a) Schreiben Sie eine Prozedur `list-length: (listof X) -> number`, die eine Liste konsumiert und die Länge der Liste als Ergebnis produziert.

      Hinweis: Die leere Liste enthält keine Elemente. Verwenden Sie Rekursion, um das Ergebnis für nicht-leere Listen zu berechnen.
   b) Schreiben Sie eine Prozedur `contains?: (listof symbol) symbol -> boolean`, die eine Liste von Symbolen und ein Symbol konsumiert und **true** produziert, wenn das Symbol in der Liste enthalten ist, sonst **false**.

      Hinweis: Die leere Liste enthält das Symbol sicher nicht. Verwenden Sie Rekursion, um das Ergebnis für nicht-leere Listen zu berechnen. Verwenden Sie dazu die Prozedur **symbol**=?: **symbol symbol** −> boolean, um zu überprüfen, ob zwei Symbole identisch sind.
   c) Die Prozedur `remove-duplicates: (listof symbol) -> (listof symbol)` soll eine Liste von Symbolen konsumieren und eine Liste ohne Duplikate produzieren. Entwickeln Sie eine passende Implementierung. Sie können auf bereits implementierte Prozeduren zurückgreifen.

      Beispiel: (remove−duplicates '(a a b)) −> '(a b).
      Verwenden Sie Rekursion und betrachten Sie drei Fälle:

- Die leere Liste.
- Das erste Element der Liste kommt im Rest der Liste vor.
- Das erste Element kommt nicht im Rest der Liste vor.

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal 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 Portal auch für sie bewertet werden kann.

Abgabedatum: Freitag, 06. 11. 2009, 16:00 Uhr

Denken Sie bitte daran, Ihren Code hinreichend gemäß den Vorgaben zu kommentieren (Scheme: Vertrag, Beschreibung und Beispiel sowie zwei Testfälle pro Funktion; 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 „Anfänger mit Listen-Abkürzungen".

5 EMail Adresse (0 Punkte)

Lesen Sie entweder regelmäßig die EMails in ihrer RBG-Mailbox oder richten Sie sich eine Weiterleitung ein. Sollten Sie Probleme damit haben, folgen Sie den Hinweisen unter

`http://www.rbg.informatik.tu-darmstadt.de/index.php?id=560#forward`

6 Kursverwaltung für Studierende (6 Punkte)

In dieser Aufgabe soll eine vereinfachte Kursverwaltung für Studierende erstellt werden. Gegeben seid dabei die folgende Definition einer Strukturen zur Repräsentierung von Studierenden mit Matrikelnummer und Studiengang, sowie eine Liste mit Beispielstudierenden.

```
1  ;; student: a the struct for modelling students
2  ;; id: number - the student id
3  ;; fos: symbol - the field of study, e.g. 'inf, 'ce
4  (define-struct student (id fos))
5
6  ;; create a list of some random students
7  (define some-students
8    (list
9     (make-student 13 'inf)
10    (make-student 532 'winf)
11    (make-student 352 'ce)
12    (make-student 54 'inf)
13    (make-student 256 'ist)))
```

Die Definitionen der Struktur `student` und der Liste `some-students` können Sie auch im Portal herunterladen.
Hinweis: Mit **cons**: **symbol** (listof X) -> (listof X) können Sie einer Liste ein Symbol voranstellen.
Hinweis: Vergessen Sie nicht Vertrag, Beschreibung und zwei Beispiele für alle Prozeduren anzugeben!

6.1 Zählen der Studierenden in einem Studiengang (2 Punkte)

Schreiben Sie eine Prozedur count−students−for: (listof student) **symbol** −> number, welche eine Liste von Studierenden und ein Symbol konsumiert. Die Prozedur produziert die Anzahl der Studierenden mit dem Studiengang **symbol**.

6.2 Verteilung der Studierenden auf Studiengänge (2 Punkte)

Schreiben Sie die Prozedur count−student−dist: (listof student) (listof **symbol**)−> (listof number). Die Prozedur konsumiert eine Liste von Studierenden und eine Liste von Studiengängen (codiert als Symbol). Sie produziert eine Liste mit der Anzahl der Studierenden des jeweiligen Studiengangs in der gleichen Reihenfolge wie die Eingabeliste.

6.3 Ausgabe der Studienfachverteilung (2 Punkte)

Implementieren Sie eine Prozedur print −student−**list**: (listof student) (listof **symbol**)−> string. Die Prozedur konsumiert eine Liste von Studierenden und von Studiengängen (codiert als Symbol). Als Ergebnis wird ein String produziert entsprechend dem folgenden Beispiel:
(print −student−**list** some−students '(inf winf ce ist)) ergibt " inf : 2 winf : 1 ce: 1 ist : 1 "
Hinweis: Beachten Sie die Nutzung von Leerzeichen hinter dem Doppelpunkt und nach jeder Zahl; das Leerzeichen am Ende ist gewollt!
Nutzen Sie dazu die folgenden primitiven Prozeduren:

- string −**append**: string∗ string −> string fügt mindestens zwei (dafür steht der ∗ bei dem ersten Parameter) Strings zu einem neuen String zusammen
- **symbol**−>string: **symbol** −> string wandelt ein Symbol in einen entsprechenden String, d.h. (**symbol**−>string 'hello) = " hello"
- number−>string: number −> string wandelt eine Zahl in den entsprechenden String: (number−>string 42)= "42"