Grundlagen der Informatik 1

Sommersemester 2011

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

Übung 2 Version: 1.1 25. 04. 2011

---

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 (one two))
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−two (**make**−my−pair 1 2))(my−pair−one (**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 eine 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 my−len: (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 der Liste kommt *nicht* im Rest der Liste vor.

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, 06. 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 „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/onlinehilfe/e-mail/#forward`

6 Wahl-o-Mat (6 Punkte)

"Wer die Wahl hat. . . "
In dieser Aufgabe implementieren wir eine Racket-basierte (und etwas vereinfachte) Fassung des *Wahl-o-Mat* der Bundeszentrale für politische Bildung (`http://www.wahl-o-mat.de`).
Der Wahl-o-Mat stellt dem Benutzer eine Reihe von Fragen zu politischen Themen und wertet dann aus, wie gut die Positionen der Parteien zur Meinung des Nutzers passen. Am Ende wird dann die „beste" Partei empfohlen[1]. Diese Aufgabe vereinfacht diese Anwendung natürlich etwas auf das GdI 1-Niveau, deutet aber an, wie das Ganze funktioniert.
Die Aufgabe nutzt die folgenden Definitionen (hier nur im Auszug; in der Vorlage finden sich auch vordefinierte Beispiele) von *politischen Einstellungen* (Struktur position) und *Parteimeinungen* (Struktur party). Eine *Einstellung* besteht dabei aus einem Thema (topic) und der *Haltung dazu*, die einen der folgenden Werte annimmt:

-2 Diese Codierung steht für eine deutlich ablehnende Haltung.

-1 Die Partei oder Person ist überwiegend gegen das Thema.

0 Die Partei oder Person ist unentschlossen oder hat keine eigene bekannte Haltung zum Thema.

[1] Ich empfehle sehr, das selbst einmal auszuprobieren, wenn eine Wahl—egal, wo—bevorsteht. Oft ist das Ergebnis durchaus anders, als man erwartet hätte, wenn man glaubt, dass man schon „die richtige Partei gefunden hat"!

1 Diese Codierung steht für eine überwiegend positive Einstellung.

2 Die Partei oder Person stimmt dem Thema eindeutig zu.

Eine Partei besteht aus dem Namen und einer (möglicherweise leeren) Liste von *Einstellungen*, wie in der Definition zu sehen ist.

```
;; position defines the attitude towards a given topic
;; topic: String − the topic
;; attitude: number − −2 (strongly against it), −1 (weakly negative),
;; 0 (undecided), 1 (weakly positive), or 2 (strongly positive)
(define−struct position (topic attitude))

;; party represents the set of values that a
;; given political party has
;; name: String − the name of the party
;; positions: (listof position) − the positions
;; (see above) the party has on different topics
(define−struct party (name positions))

;; example elements
(define klaus−klammer (list (make−position "A" 2)
                            (make−position "B" −1)))
(define klammer−partei
  (make−party "Klammern"
              (list (make−position "A" 2)
                    (make−position "B" 0)
                    (make−position "C" 1))))
```

Die Definitionen der Strukturen position und party können Sie, zusammen mit Beispieldefinitionen, im Kurs zur Vorlesung herunterladen.
Hinweis: Vergessen Sie nicht Vertrag, Beschreibung, Beispiel und zwei Tests für alle Prozeduren anzugeben!

6.1 Bestimmen der Haltung zu einem Thema (1,5 Punkte)

Schreiben Sie eine Prozedur extract−attribute−for: **String** (listof position)−> **number**. Diese konsumiert ein Thema (codiert als String) und eine Liste von position-Strukturen. Als Ergebnis soll entweder der in der Liste abgelegte Wert zu diesem Thema zurückgegeben werden oder—falls kein passender Eintrag existiert—0.
Zu beachten: Sollte die Liste mehrere Einträge zu dem gesuchten Thema enthalten, ist der *erste* zu verwenden.
Hinweis: Um den Schreibaufwand etwas zu reduzieren, wird hier extra *keine* Instanz von party übergeben.

6.2 Bewertung der Einstellung einer Partei (2,5 Punkte)

Schreiben Sie eine Prozedur party−penalty: party (listof position)−> **number**. Diese Prozedur konsumiert eine Partei-Struktur sowie eine Liste von Einstellungen des Wählers und bewertet die Partei anhand der Übereinstimmung der jeweiligen Haltungen. Dabei soll für jedes Thema in der Liste der dem Wähler wichtigen Themen die Position der Partei bestimmt werden. Ist die Position der Partei identisch, wird dies mit 0 bewertet, andernfalls wird der *Betrag des Unterschieds*—der zwischen 1 und 4 liegt—in den Meinungen mit 3 multipliziert.
Beispiel: Partei A ist stark für Thema „X“ (+2) und stark gegen Thema „Y“. Der aktuell betrachtete Wähler ist ebenfalls stark für Thema „X“, hat aber für Thema Y explizit „keine eindeutige Meinung“—er hat in diesem Fall also die Bewertung 0 für Thema Y. Der Unterschied beträgt bei Thema „X“ 0 (beide +2), bei Thema „Y“ hingegen 2 (-2 zu 0). Die Bewertung der Partei ist der jeweils dreifache *Betrag* der Unterschiede, hier also $3 \times 0 + 3 \times |-2| = 6$.

Zu beachten: Würde das Thema „Y“ nicht in der Liste des Wählers auftauchen, ist es auch nicht zu berücksichtigen—auch nicht mit dem Wert 0. Hier unterscheidet sich also der Wähler von den Parteien: Wähler können Themen einfach ignorieren („auslassen“). Diese werden dann vom Wahlomat ignoriert, da sie für den Wähler und seine Wahlentscheidung ja nicht relevant sind. Wenn der Wähler hingegen zu einem Thema unschlüssig ist, interessiert ihn das Thema prinzipiell durchaus; Parteien, die eine starke Haltung zu dem Thema haben, passen daher vielleicht nicht gut zum Wähler und werden daher „bestraft“.
Das Ergebnis der Prozedur ist also genau dann 0, wenn die Meinungen von Wähler und Partei bei den bei beiden vertretenen Themen genau identisch sind; in allen anderen Fällen entsteht ein Ergebnis *größer als 1*.
Hinweis: Die Listen der Einstellungen von Wähler und Partei sind voneinander unabhängig. Sie können *weder* unterstellen, dass die Listen die gleiche Länge haben noch dass sich Wähler und Partei beide zu genau den gleichen Themen geäußert haben!

6.3 Bewertung aller Parteien (2 Punkte)

Implementieren Sie nun noch eine Prozedur party−penalties: (listof party)(listof position)−> (listof evaluation). Diese bestimmt zu jeder Partei anhand der Positionen des Wählers das Endergebnis. Das Endergebnis jeder einzelnen Partei ist jeweils vom Typ evaluation und wird wie folgt definiert:

```
1 ;; an evaluation is either empty or a (list n p), with
2 ;; n: String − the name of a party
3 ;; p: number − the penalty for a party for a given list of positions
4 ;; example: (list "Party A" 12) is a valid evaluation that gives
5 ;; the party called "Party A" a penalty of 12
```

Die Prozedur party−penalties konsumiert eine Liste von Parteien und die Positionen des Wählers (als Liste von position-Strukturen). Als Ergebnis wird eine Liste von evaluation-Strukturen produziert, die genauso lang wie die Liste der Parteien ist und zu jeder Partei in der übergebenen Reihenfolge aus einer Liste mit dem Namen der Partei und der Bewertung der Partei besteht.
Beispiel: wenn es zwei Parteien "Partei A" und "Partei B" gibt und deren Bewertung 3 bzw. 9 ist, so lautet das Ergebnis von (party−penalty ...)= (**list** (**list** "Partei A" 3)(**list** "Partei B" 9)).