1 Generalisierte Variablen (1 + 3 + 3 + 3 Punkte)

a) Lies die (relevanten Teile der) Dokumentation zu den neu eingeführten Common-Lisp Funktionen (siehe '`http://www.dfki.de/~becker/lispkurs/clm/new-index.html`'). Verwende den Index, um die Funktionen `rplacd` und `nconc` kennenzulernen bzw. zu wiederholen.

b) Erkläre die Auswirkungen (inklusive möglicher Seiteneffekte) der folgenden Anweisungen; am besten unter Verwendung der Kästchenschreibe für `cons`-Zellen:

```
? (defparameter *foo* (list 1 2 3))
→ *FOO*
? (setf (first (rest *foo*)) 42)
→ 42
```

Der Zeiger, der aus dem 'car' der zweiten Zelle von `*foo*` zeigt, wird „verbogen" und zeigt statt auf 2 nun auf 42. Als Seiteneffekt ändert sich also der Wert von `*foo*` auf `(1 42 3)`.

```
(defparameter *foo* (list 1 2 3))

*foo*--->[ | ] /->[ | ] /->[ | ] /->NIL
          | |  |   | |  |   | |  |
          | \--/   | \--/   | \--/
          v        v        v
          1        2        3

(setf (first (rest *foo*)) 42)

*foo*--->[ | ] /->[ | ] /->[ | ] /->NIL
          | |  |   | |  |   | |  |
          | \--/   \ \--/   | \--/
          v         \      v
          1        2 \      3
                      |
                      v
                      42
```

c) Schreibe eine zweistellige, rekursive Funktion `append2`, die ihre zwei Eingabelisten ähnlich wie das vordefinierte `append` konkateniert; allerdings soll `append2` völlig ohne Aufruf von `cons` oder anderer Funktionen, die neue `cons`-Zellen erzeugen, auskommen, z.B.

```
? (append2 '(1 2 3) '(4 5 6))
→ (1 2 3 4 5 6)
```

Hinweis: Die Konkatenation soll durch destruktive Veränderung (z.B. mit `setf`, `nconc` oder `rplacd`) der ersten Eingabeliste realisiert werden.

```
(defun append2 (list1 list2)
  (cond ((null list1) list2)
((null list2) list1)
(T (setf (cdr (last list1)) list2)
   list1))) ;; Achtung: SETF w"urde LIST2 zur"uckgeben.
```

d) Diskutiere den Effekt von `append2` anhand der Kästchendarstellung des obigen Beispiels. Gib geeignete Beispielaufrufe an, die zeigen, daß die Veränderung der ersten Eingabeliste dauerhaft ist.

```
(setq a '(a b) b '(x y))

  a  --->[ | ] /->[ | ] /->NIL
          | |  |    | |  |
          | \--/    | \--/
          v         v
          A         B

  b  --->[ | ] /->[ | ] /->NIL
          | |  |    | |  |
          | \--/    | \--/
          v         v
          X         Y

(append2 a b)

  (last a) -------\
                  |
                  v
  a  --->[ | ] /->[ | ]
          | |  |    | |
          | \--/    | \--\
          v         v    |
          A         B    |
                         |
         /--------------/
         |
         v
  b  --->[ | ] /->[ | ] /->NIL
          | |  |    | |  |
          | \--/    | \--/
          v         v
          X         Y

USER(133): (setq a '(a b) b '(x y))
(X Y)
USER(134): (append2 a b)
(A B X Y)
USER(135): a
(A B X Y)
```

2 Verbmorphologie (4 + 3 + 3)

a) Modifiziere die Funktion `token2code` so, daß sie als Alphabet alle alphanumerischen Zeichen (aus dem Bereich `char-code` >= 0 und < 256) auf Zahlen von 2 aufwärts abbildet (die ersten beiden Spalten der Übergangstabelle sind reserviert). Anders als in der Kodierung mit `case` soll `token2code` nunmehr konstante Komplexität haben. Führe weiterhin eine globale Variable `*code-maximum*` ein, die den größtmöglichen Rückgabewert für `token2code` enthält (d.h. den größten Spaltenindex).

Einfachste Lösung, die aber sehr große Tabellen erzeugt:

```
(defun token2code (char)
  (when (alphanumericp char)
    (+ 2 (char-code char))))
(defparameter *code-maximum* (+ 255 2))

?    (token2code #\s)
-->  117
```

Bessere Lösung, die mit wesentlich kleinerem Wert für `*code-maximum*` auskommt, und daher kleinere Tabellen erzeugt:

```
(defparameter *tokens2codes* (make-array 256))

(defparameter *code-maximum*
  (loop with code-for-token = 1    ;; first call to incf makes it 2
        for i from 0 to 255 by 1
        when (alphanumericp (code-char i))
        do (setf (aref *tokens2codes* i) (incf code-for-token))
        finally (return code-for-token)))

(defun token2code (token) (aref *tokens2codes* (char-code token)))

?    (token2code #\s)
-->  56
```

b) Schreibe eine einstellige Funktion `string2table`, die aus einer Zeichenkette einen Automaten (mit dem Alphabet aus Aufgabenteil (a)) erzeugt, der die übergebene Zeichenkette und weiter nichts erkennt, z.B.

```
? (traverse (string2table "denk") "denk")
→ (4)

(defun string2table (string)
  (loop with length = (length string)
with table = (make-array (list (1+ length) (1+ *code-maximum*)))
for i from 0 below length
do (setf (aref table i (token2code (schar string i)))
 (list (1+ i)))
finally (setf (aref table length 0) string)
                (return table)))
```

c) Extrahiere aus dem Automaten aus Aufgabe (2) des ersten Übungsblattes den Teil, der die Suffixe der deutschen Verbmorphologie (im Präsens, Indikativ, Aktiv) erkennt; erweitere den Automaten für das in (a) angegebene Alphabet und ändere die Funktion `traverse` dahingehend, daß als Rückgabewert nicht nur '`t`' oder '`nil`' sondern ein beliebiger Wert geliefert wird, der mit dem erreichten Endzustand assoziiert ist, z.B.

```
? (traverse *praesens* "st")
→ (2sg)

(defparameter *praesens*
  (let ((table (make-array (list 6 (+ *code-maximum* 1)))))
    (setf (aref table 0 (token2code #\e)) '(1))
    (setf (aref table 1 0) '1sg)
    (setf (aref table 0 (token2code #\s)) '(3))
    (setf (aref table 0 (token2code #\t)) '(5))
    (setf (aref table 5 0) '3sg-or-2pl)
    (setf (aref table 1 (token2code #\n)) '(2))
    (setf (aref table 2 0) '1pl-or-3pl)
    (setf (aref table 3 (token2code #\t)) '(4))
    (setf (aref table 4 0) '2sg)
    table))

(defun traverse (table input &optional
                             (state 0)
                             (position 0)
                             (n (length input)))
  (if (= position n)
    (when (aref table state 0) (list (aref table state 0)))
    (loop with forward = (1+ position)
  for next in (aref table state
    (token2code (schar input position)))
  nconc (traverse table input next forward n))))
```

Abgabetermin war Freitag, der 5. Mai 2000