6. Übungsblatt zu Programmiersprachenkonzepte

Manuel Schwarz, Michael Stypa

12. Januar 2011

Aufgabe 6.1

1.
```
(defun mycount (item liste)
    ;;fuer den Fall, dass liste eine Liste ist
    (cond ((listp liste)
           (cond
           ((null liste) 0)      ;;Rekursionsanker
           ;;wenn item == 1. Element,
           ;;dann 1 zurueckgeben und
           ;;mycount mit dem Rest aufrufen
           ((eq item (first liste))
            (+ 1 (mycount item (cdr liste)))
           )
           ;;sonst nur mycount mit dem Rest aufrufen
           (T (mycount item (cdr liste)))
           ))
           ;;fuer den Fall, dass liste ein String ist
           ((stringp liste)(cond
           ((null liste) 0)    ;;Rekursionsanker
           ;;String zur Liste aus Charactern machen,
           ;;dann s.o.
           ((eq item (first (coerce liste 'list)))
            (+ 1 (mycount item (cdr (coerce liste 'list)))))
           (T (mycount item (cdr (coerce liste 'list))))
       )
     )
   )
 )
```

2.
```
(defun mycount-if (predicate liste)
    ;;fuer den Fall, dass liste eine Liste ist
    (cond ((listp liste)
           (cond
           ((null liste) 0) ;;Rekursionsverankerung
           ;;Methode in predicate angewandt auf das 1.Element aus liste
           ((apply predicate (list (first liste)))
```

```
            (+ 1 (mycount-if predicate (cdr liste))))
            (T (mycount-if predicate (cdr liste)))
            ))
            ;;fuer den Fall, dass liste ein String ist
        ((stringp liste)(cond
            ((null liste) 0)
            ((apply predicate (list(first (coerce liste 'list)))) (+ 1
                                (mycount-if predicate (cdr (coerce liste 'list)))))
            (T (mycount-if predicate (cdr (coerce liste 'list))))
          )
        )
      )
    )
```

Aufgabe 6.2

```
(defun matmulmat (m1 m2)
  (setf dim1 (array-dimensions m1))
  (setf dim2 (array-dimensions m2))
  (setf ergm (make-array (list (first dim1) (second dim2)) :initial-element 0))
  (if (not (= (first dim1) (second dim2)))
      (error "Dimensionen passen nicht!")
      (dotimes (zeile (first dim1) ergm)
               (dotimes (spalte (second dim2))
                        (setf (aref ergm zeile spalte)
                              (do ((i 0 (+ i 1)) (sum 0))
                                  ((= i (first dim2)) sum)
                                  (setf sum (+ sum (* (aref m1 zeile i)
                                                      (aref m2 i spalte))))
                              )
                        )
               )
      )
  )
)
```

Aufgabe 6.3

```
(defun quadmatp (m)
  (setf dim (array-dimensions m))
  (eq (first dim) (second dim))
)

(defun ematp (m)
  (setf dim (array-dimensions m))
  (setf err 0)
```

```
  (if (quadmatp m)
      (dotimes (zeile (first dim))
               (dotimes (spalte (second dim))
                        (setf wert (aref m zeile spalte))
                        (if (= zeile spalte)
                            (if (not (= wert 1))
                                (setf err 1)
                            )
                            (if (not (= wert 0))
                                (setf err 1)
                            )
                        )
               )
      )
      (error "Dimensionen passen nicht!")
  )
  (if (= err 1)
      NIL
      T
  )
)

(defun transpm (m)
  (setf dim (array-dimensions m))
  (setf newm (make-array (list (second dim) (first dim))))
  (dotimes (zeile (first dim) newm)
           (dotimes (spalte (second dim))
                    (setf (aref newm spalte zeile) (aref m zeile spalte))
           )
  )
)

(defun orthom (m)
  (ematp (matmulmat m (transpm m)))
)
```