Übungsblock 7

Lösungen Übungsblatt 6:

Aufgabe 1: Multiplikation zweier Matrizen

```
(defun matmulmat (m1 m2)
    (setf d1 (array-dimensions m1))
    (setf d2 (array-dimensions m2))
    (setf ergm (make-array (list (first d1) (second d2))))
    (if (not (= (second d1)  (first d2)))
        (error "Dimensionen passen nicht")
       (dotimes (i (first d1) T)
                  (dotimes (k (second d2) T)
                              (setf sum 0)
                              (setf (aref ergm i k) (dotimes (j (second d1) sum)
                                          (setf sum (+ sum (* (aref m1 i j) (aref m2 j k))))
    )))))
    ergm
)
```

Übungsblock 7-1

Aufgabe 2.1:

```
(defun quadmatp (m)
    (if (and (= 2 (array-rank m))
             (= (array-dimension m 0) (array-dimension m 1)))
        T          ;then-Teil
        NIL ;else-Teil
    )
)
```

Übungsblock 7-1

Aufgabe 2.2:

```
(defun ematp (m)
(if (quadmatp m)
      (do       ((i 0 (+ i 1))
                 (flag 0))
                ( (= i (array-dimension m 0)) T)
                (do ((j 0 (+ j 1)))
                    ( (= j (array-dimension m 1)) T)
                    (cond ((= i j) (if (/= (aref m i j) 1) (return-from ematp NIL)))
                          (T       (if (/= (aref m i j) 0) (return-from ematp NIL )))
                )
       )
    )
    NIL         ;else-Teil, m ist nicht quadratisch
))
```

Übungsblock 7

Aufgabe 2.3:

```
(defun transpm (m)
  (setf ergm (make-array (array-dimensions m)))
  (if  (not (quadmatp m)) (error "Die Matrix ist nicht quadratisch!")
   (dotimes (i (array-dimension m 0) T)
       (dotimes (j (array-dimension m 1))
                  (setf (aref ergm i j) (aref m j i))
       )
   ))
  ergm
)
```

Aufgabe 2.4:

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

Übungsblock 7

Aufgabe 3.1:

```
(defstruct student   matrnr vor nach fach geschlecht)
```

Aufgabe 3.2:

```
(defun save (datei st)
      (with-open-file (stream datei :direction :output
                              :if-does-not-exist :create
                              :if-exists :append)
            (if (streamp stream)
                  (format stream "~&~6D ~15A ~20A ~20A ~1A~%"
                  (student-matrnr st) (student-vor st) (student-nach st)
                  (student-fach st) (student-geschlecht st))
                  (error "Konnte keinen stream erzeugen") ; else-Teil
            ))
)  ; ende defun save
```

Übungsblock 7

Aufgabe 3.3:

```
(defun loadstudis (datei)
    (let ((stream (open datei :direction :input
                          :if-does-not-exist :error))
          (liste ())
        )
        (do ((matrnr (read stream NIL NIL) (read stream NIL NIL))
             (vor (read stream NIL NIL) (read stream NIL NIL))
             (nach (read stream NIL NIL) (read stream NIL NIL))
             (fach (read stream NIL NIL) (read stream NIL NIL))
             (geschlecht (read stream NIL NIL) (read stream NIL NIL)))
          ((or (eq matrnr NIL) (eq vor NIL) (eq nach NIL)
               (eq fach NIL) (eq geschlecht NIL))
             (close stream)              ;Aufraeumarbeiten:
             liste                            ;Resultat:
          )
    …
```

Übungsblock 7

Aufgabe 3.3 (Fortsetzung):

```
…
                ;do-Schleifenrumpf:
                (push (list matrnr vor nach fach geschlecht) liste)
              ) ;ende do
        ); ende let
) ; ende loadstudis
```

Übungsblock 7

Aufgabe 3.4:

```
(defun geschlechtfach (datei fach)
  (let (  (liste (loadstudis datei))
          (anzw 0)
          (anzm 0))
      (do ( (st (car liste) (car liste)) )
           ((eq st NIL) (format T "~%~A studieren ~D Frauen und ~D Maenner"
                                                                  fach anzw anzm) T)
            (if (not (eql (find fach st) NIL))
                   (if (equal (car (last st)) 'm)
                          (setf anzm (+ 1 anzm))
                          (setf anzw (+ 1 anzw))))
            (setf liste (cdr liste))
      )
   ))
```

Übungsblock 7

Aufgabe 3.5:

```
(progn
   (load "student.lisp")
   (save "studenten.txt" (make-student :matrnr 123456
                                       :vor "Max" :nach "Muster " :fach "Bio" :geschlecht "m"))
   (save "studenten.txt" (make-student :matrnr 223456
                                       :vor "Klaus" :nach "Kleber " :fach "Bio" :geschlecht "m"))
   . . .; hier weitere StudentInnen erzeugen und speichern
   (print (loadstudis "studenten.txt"))
   (geschlechtfach "studenten.txt" 'Informatik)
)
```