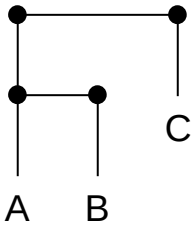
Übung 12
Musterlösung

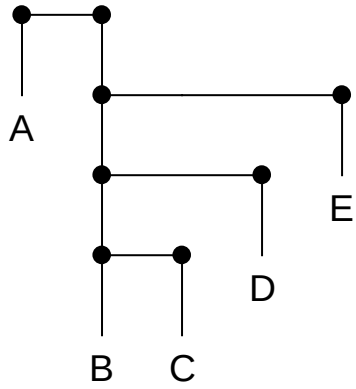
Aufgabe 12.1

a) Speicherstruktur

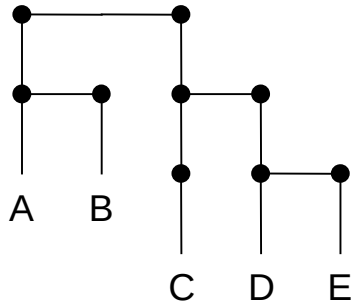
Liste 1



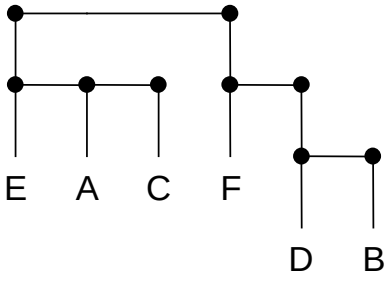
Liste 2



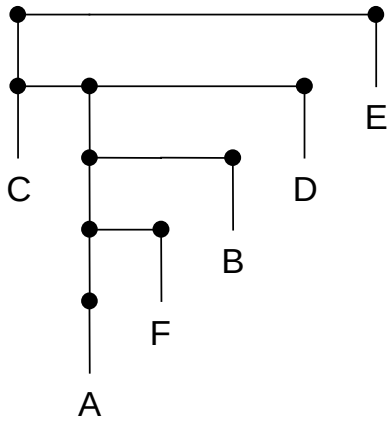
Liste 3



Liste 4



Liste 5



b) Aufbau der Listen aus den Atomen und leeren Listen mittels CONS

```
(setq liste1 (cons (cons 'A (cons 'B ()))
                   (cons 'C ()) ) )

(setq liste2 (cons 'A (cons (cons (cons (cons 'B (cons 'C ()))
                                        (cons 'D ()))
                                  (cons 'E ()))
                            ()) ) )

(setq liste3 (cons (cons 'A (cons 'B ()))
                   (cons (cons (cons 'C ())
                               (cons (cons 'D (cons 'E ()))
                                     ()) )
                         ()) ) )

(setq liste4 (cons (cons 'E (cons 'A (cons 'C ())))
                   (cons (cons 'F (cons (cons 'D (cons 'B ()))
                                        ()) )
                         ()) ) )

(setq liste5 (cons (cons 'C (cons (cons (cons (cons 'A ())
                                              (cons 'F ()))
                                        (cons 'B ()))
                                  (cons 'D ())) )
                   (cons 'E ()) ) )
```

c) Zerlegen der in Teil b) gebauten Liste in die neue Liste (A B C)

```
(setq resultat1 (cons (car (car liste1))
                      (cons (car (cdr (car liste1)))
                            (cdr liste1)) ) )

(setq resultat2 (cons (car liste2)
                      (car (car (car (cdr liste2))))) )

(setq resultat3 (cons (car (car liste3))
                      (cons (car (cdr (car liste3)))
                            (car (car (cdr liste3)))) ) )

(setq resultat4 (cons (car (cdr (car liste4)))
                      (cons (car (cdr (car (cdr (car
                                (cdr liste4))))))
                            (cdr (cdr (car liste4)))) ) )

(setq resultat5 (cons (car (car (car (car (cdr (car liste5))))))
                      (cons (car (cdr (car (cdr (car liste5)))))
                            (cons (car (car liste5))
                                  ()) ) ) )
```

Aufgabe 12.2

a) Funktion `wurzel`:

```
;; Hilfsfunktion
(defun quadrat (x)
    (* x x) )


;; berechnet die Wurzel von a
(defun wurzel (a)

    ; ermittelt Wurzel der Zahl a durch Probieren von 1 aufwaerts
    (defun ermittleWurzel (a zaehler)
        (cond ((< (quadrat zaehler) a)
                  (ermittleWurzel a (+ zaehler 1)))
              ((= (quadrat zaehler) a) zaehler)
              (T 0)) )

    ; starte Probe mit 1
    (ermittleWurzel a 1))
```

Probelauf der Funktionen `quadrat` und `wurzel`:

```
> (quadrat 2)
4
> (quadrat 5)
25
> (wurzel 9)
3
> (wurzel 7)
0
> (wurzel 25)
5
```

b) Prozedur `pythagoras_tripel`:

```
;; die Wurzelfunktion aus Teil a) wird hier verwendet

;; Hilfsfunktion, berechnet die Summe der Quadrate von x und y

(defun quadratsumme (x y)
    (+ (quadrat x) (quadrat y)))
```

```
;; ermittelt durch Probieren aller (u, v)-Kombinationen im
;; Wertebereich (a..b) eine passende Zahl w und gibt bei Erfolg
;; das pythagoraeische Tripel (u, v, w), andernfalls NIL zurueck

(defun pythagoras_tripel (a b)

    ;; Hilfsfunktion, probiert alle Kombinationen von (u, v)
    (defun probiere (u v)
        (cond ((= (wurzel (quadratsumme u v)) 0)
               ; Tripel noch nicht gefunden
               (cond ((and (= u b) (= v b)) ()) ;leere Liste zurueck
                     ((and (< u b) (= v b)) (probiere (+ u 1) a))
                     ((< v b)               (probiere u (+ v 1)))))

              ; passendes Tripel gefunden, als Liste zurueckgeben
              (T (list u v (wurzel (quadratsumme u v))))) )

    ; starte Probe mit u = a und v = a
    (probiere a a))
```

Probelauf der Funktion `pythagoras_tripel`:

```
> (pythagoras_tripel 1 5)
(3 4 5)
> (pythagoras_tripel 3 4)
(3 4 5)
> (pythagoras_tripel 3 5)
(3 4 5)
> (pythagoras_tripel 4 5)
NIL
> (pythagoras_tripel 5 10)
(6 8 10)
> (pythagoras_tripel 10 20)
(12 16 20)
> (pythagoras_tripel 15 16)
NIL
```

Aufgabe 12.3

Prozedur `nimmspiel`:

```
;; das Nimmspiel erwartet als Eingabe die Anzahl der Staebchen
;; zu Beginn des Spiels und spielt anschliessend gegen den
;; Benutzer mittels eines PRINT/READ-Dialogs

(defun nimmspiel (n)
    ; drucke Anzahl der Staebchen im Spiel
    (print (cons n '(Staebchen)))
    ; Benutzer darf das erste Staebchen nehmen!
    (cond ((or (> n 1))
              ; frage Benutzer nach seinem Zug
              (print '(Wieviele Staebchen nehmen Sie ?))
              (let ((n (- n (read))))
                 (print (cons n '(Staebchen)))
                 (cond ((> n 1)
                           (let ((zug (rem (+ n 3) 4)))
                                (cond ((= zug 0)
                                    (nimmspiel (- n 1)))
                                      ( T
                                    (nimmspiel (- n zug))))))
                       ((= n 1) (print '(Gratuliere !))))))

          ; der Benutzer hat keine Chance mehr!
          ((= n 1)(print '(Ich habe gewonnen !)))))
```

Alternative mit SETQ statt LET:

```
;; Alternative zur obigen Implementierung, die SETQ statt
;; LET verwendet die Funktion arbeitet genauso wie oben,
;; allerdings haben die mit SETQ gesetzten Variablen n und
;; zug nicht nur lokale Gueltigkeit wie bei LET

(defun alt_nimmspiel (n)
    ; drucke Anzahl der Staebchen im Spiel
    (print (cons n '(Staebchen)))
    ; Benutzer darf das erste Staebchen nehmen!
    (cond ((or (> n 1))
              ; frage Benutzer nach seinem Zug
              (print '(Wieviele Staebchen nehmen Sie ?))
              (setq n (- n (read)))
              (print (cons n '(Staebchen)))
              (cond ((> n 1)
                        (setq zug (rem (+ n 3) 4))
                        (cond ((= zug 0) (nimmspiel (- n 1)))
                              (T     (nimmspiel (- n zug)))))
                    ((= n 1) (print '(Gratuliere !)))))

          ; der Benutzer hat keine Chance mehr!
          ((= n 1)(print '(Ich habe gewonnen !)))))
```

Probeläufe der Prozedur `nimmspiel`:

```
> (nimmspiel 15)

(15 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 2

(13 STAEBCHEN)
(12 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 1

(11 STAEBCHEN)
(9 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(6 STAEBCHEN)
(5 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(2 STAEBCHEN)
(1 STAEBCHEN)
(ICH HABE GEWONNEN !)
(ICH HABE GEWONNEN !)
> (nimmspiel 15)

(15 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 2

(13 STAEBCHEN)
(12 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(9 STAEBCHEN)
(8 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(5 STAEBCHEN)
(4 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(1 STAEBCHEN)
(GRATULIERE !)
(GRATULIERE !)
```

Die Strategie bei diesem Spiel ist, bei jedem Zug so viele Stäbchen zu nehmen, dass die verbleibende Anzahl Stäbchen eins größer ist als die nächste durch vier teilbare Zahl. Gelingt dem Spieler ein solcher Zug, dann hat er das Spiel bereits gewonnen. Das Problem bei diesem Spiel ist, dass wenn auch der Gegner diese Strategie kennt, immer derjenige gewinnt, der die Strategie zuerst umsetzen kann.