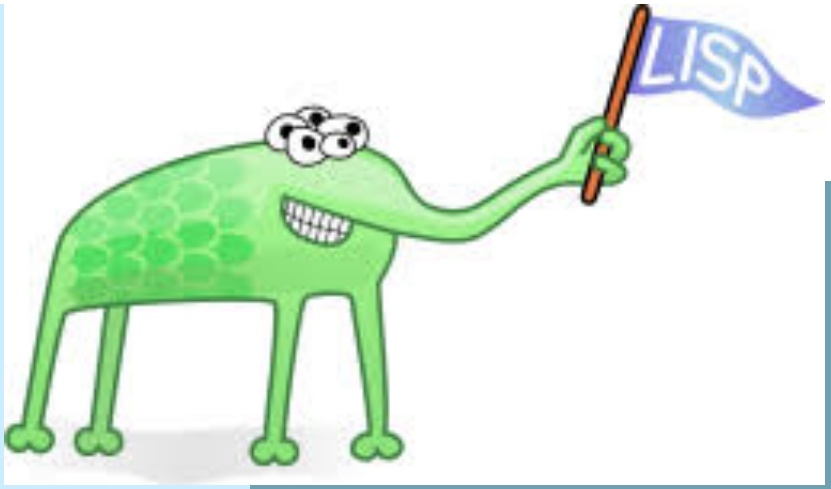
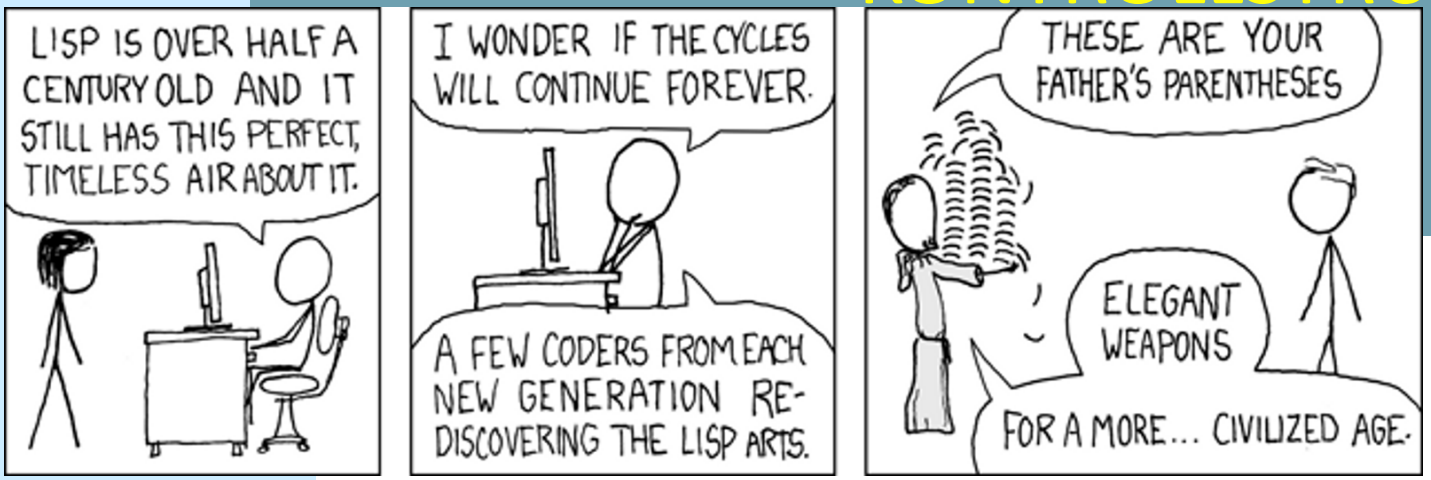
8. Kapitel



LISP
TEIL 4
KONTROLLSTRUKTUREN



Programmieren 3

Prof. Dr. Wolfgang Schramm

Kontrollstrukturen (vgl. Dijkstra)

- Sequenzen
- Bedingungen
- Iteration
- Rekursion

Für Verzweigungspunkte und Wiederholungen

Sequenz


```
{ <s-expr>   | <form> }*
```

- Bei der Definition einer Funktion kann eine Sequenz von symbolischen Ausdrücken angegeben werden.
- Das Ergebnis der Funktion ist der Wert des letzten symbolischen Ausdrucks.

Beispiel:

```
(defun foo (x)
   (+ x x)
   (oddp x)
)

> (foo 3) ⟼    T
```

Iteration – do-Schleife


```
(do ( {( var [init [step]] ) }* )
    ( end-test {result}* ) {form}* )
```
- *Makro*

- Allgemeines Iterationskonstrukt, bei dem eine beliebige Anzahl lokaler Lauf-Variablen genutzt werden.
- Diese **Variablen** sind **innerhalb der Iteration gebunden** und können bei jedem Iterationsschritt wie gewünscht verändert werden.
  a) Initialisierung der Laufvariablen var mit init bzw. nil.
  b) Die Wertzuweisung erfolgt parallel.
  c) Abbruchtest und gegebenenfalls die erste Iteration ({form}*).
  d) Iterationsschritt: die Variablen erhalten gleichzeitig den jeweils durch step bestimmten Wert bzw. sie bleiben unverändert falls step nicht spezifiziert wurde.
- `do*` - Schleife: unterscheidet sich von einer DO-Form allein dadurch, dass die Wertzuweisung bei der Initialisierung und Aktualisierung der Variablen sequentiell durchgeführt wird.

Iteration – dotimes-Schleife

`DOTIMES (Var Zähl-Form [Resultat]) {Form}*-`
[Makro]

- Bei der Evaluierung einer DOTIMES-Form wird zunächst Zähl-Form evaluiert.
- Diese Form muss zu einer Zahl n evaluieren.
- Anschließend wird der Anweisungsblock n-mal ausgeführt, wobei die Variable Var nacheinander an die Werte 0 bis n-1 gebunden ist.
- Wenn n <= 0, dann wird der Anweisungsblock nicht ausgeführt.
- Wenn die Resultat-Form weggelassen wird, evaluiert die DOTIMES-Form zu NIL.

Iteration – dolist-Schleife

`DOLIST (Var List-Form [Resultat]) {Form}*` - [Makro]

- Bei der Evaluierung einer DOLIST-Form wird zunächst List-Form evaluiert.
- Diese Form muss zu einer Liste evaluieren.
- Alle Elemente dieser Liste werden anschließend nacheinander, von links nach rechts vorgehend, an die Variable Var gebunden, und der Anweisungsblock wird unter Berücksichtigung dieser Bindung ausgeführt.
- Wenn die Resultat-Form weggelassen wird, evaluiert die DOLIST-Form zu NIL.

Iteration – Loop / Endlos-Schleife

`LOOP {Form}*` - *Makro*

- Die Formen werden solange immer wieder ausgeführt, bis durch eine RETURN-Form ihre Ausführung explizit terminiert wird.

`RETURN [Resultat]` - *Makro*

- Die Evaluierung einer RETURN-Form bewirkt, dass der implizit durch ein iteratives Konstrukt wie DO oder LOOP gebildete Block verlassen wird.
- Ist eine Resultat-Form spezifiziert, dann wird der Wert dieser Form zurückgegeben; sonst NIL.

Loop - Beispiele


```
(loop for i
      below 10
      sum i)

; > 25

(loop for i
      from 5
      to 10
      sum i)

; > 45
```

```
(loop for i
      in '(1 2 3 4 5)
      sum i)

; > 15

(loop for i
      below 10
      when (oddp i)
         sum i)

; > 25
```

Bedingte Anweisung (cond)

(cond { (<test> <form>*) }*)

- Es können mehrere Test-Klauseln hintereinander stehen.
- Die Klauseln werden nacheinander abgearbeitet, bis ein <test> einen Wert ungleich nil liefert.
- Danach werden die zugehörigen symbolischen Ausdrücke <form>* ausgewertet.
- Ergebnis: Wert des zuletzt evaluierten Ausdrucks.

Beispiel: L ist eine Liste

```
(cond ((null L) 'LEER)
      (T 'NICHT-LEER))   ; Ergebnis = LEER, wenn L = NIL
```

Alternative

```
(cond  (L 'NICHT-LEER)      ; Wert von L != NIL
       (T 'LEER))
```

Bedingte Anweisung (if)

`(if (test) (then) [(else)] )` - [Special Form]

a) Evaluation der test-Form.

b) Wenn das Ergebnis nicht NIL ist, wird die then-Form evaluiert;

c) Sonst (sofern vorhanden) die else-Form.

- Ergebnis: Wert der evaluierten Form wird als Wert der IF-Form zurückgegeben.

Fallunterscheidung - case

`CASE Key-Form {(({(Key}*) |Key } {Form}*)}*` - *[Makro]*

- Die CASE-FORM besteht aus einer Key-Form und einer Folge von Klauseln.
- Jede Klausel besteht aus einem Key bzw. einer Liste von Keys und einer Folge von Formen.
- Zunächst wird die Key-Form evaluiert. Dann wird die erste Klausel gesucht, deren Key mit dem Wert der Key-Form identisch ist bzw. in deren Key-Liste der Wert der Key-Form enthalten ist.
- Anschließend werden die Formen dieser Klausel ausgeführt und als Wert der CASE-Form der Wert der letzten Form zurückgegeben.
- Gibt es keine derartige Klausel, evaluiert die CASE-Form zu NIL.

- Die Standardkontrollstruktur in LISP ist das COND(itional), das aus einer Folge von Bedingungs-Aktionspaaren besteht.

Weitere Kontrollstrukturen

- `WHEN Test {Form}*` - *[Makro]*
  - Auswertung der Test-Form:
  - Liefert sie einen anderen Wert als NIL, werden die Formen ausgeführt und der Wert der letzten Form als Wert der WHEN-Form zurückgeliefert; sonst evaluiert sie zu NIL.

- `UNLESS Test {Form}*` - *[Makro]*
  - Auswertung der Test-Form:
  - Liefert sie NIL, werden die Formenausgeführt und der Wert der letzten Form als Wert der UNLESS-Form zurückgeliefert; sonst evaluiert sie zu NIL.

- `PROGN {Form}*` - *[Special Form]*
  - PROGN nimmt eine Folge von Formen als Argumente, evaluiert sie nacheinander und liefert als Wert den Wert der letzten Form.
  - Quasi eine Blockanweisung.