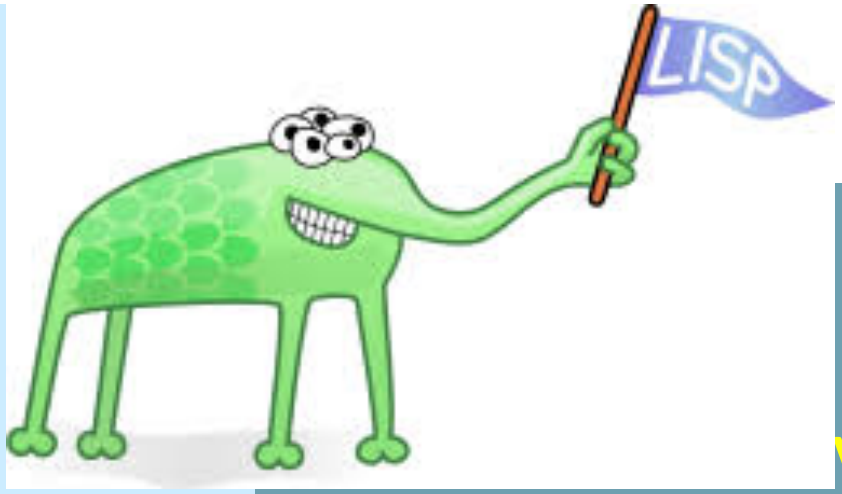
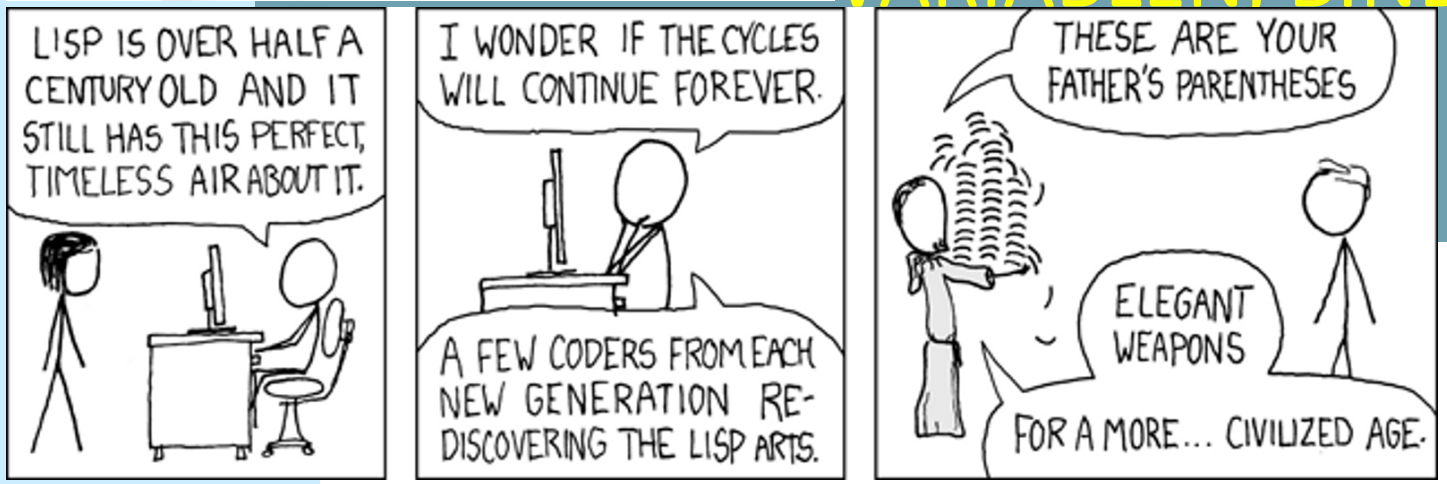
6. Kapitel



LISP
TEIL 2
VARIABLEN, BINDUNGEN



Programmieren 3

Prof. Dr. Wolfgang Schramm

Übersicht

Wertzuweisungen

- Es gibt in Common LISP eine ganze Reihe von Funktionen bzw. Special Forms.
- Special Forms, die es ermöglichen, einem Symbol einen Wert zuzuweisen (genauer: der Wert, der aktuell Bindung an eine Variablen gebunden ist, wird geändert).

Funktionen vs. Special Forms

- Unterschied zwischen *Funktionen* und *special forms* (= Spezialfall)
  $\Longrightarrow$ Behandlung der Argumentbezeichner.
- Funktionsaufruf: Argumente werden immer evaluiert.
- *special forms:* Argumente werden nicht evaluiert
  - Bsp. special form QUOTE.
  - Würde QUOTE den Argumentbezeichner evaluieren, würde das in den meisten Fällen zu der Fehlermeldung führen, die man durch Verwendung von QUOTE gerade vermeiden will.
  - `(quote a)` → `A`
  - Kurzform: `'a`

Wertzuweisungen: SETQ, PSETQ, SET

- `SETQ {var form}*` = Variablenzuweisung (von links nach rechts), Ergebnis = Wert der letzten Zuweisung - *special form*
  (Argumente werden nicht ausgewertet – sondern als Spezialfall bearbeitet)

- `PSETQ {var form}*` = parallele Zuweisung - *Makro*
  - Beispiel:

```
(setq a 1 b 2)
(psetq a b b a)
```

- `SET symbol value` *Funktion*
  - Beispiel:

```
(set (if (eq a b) 'c 'd) 'foo)
```

Wertzuweisungen: SETF

- `SETF {place newvalue}*` – *Makro*
  - `place` (= adressierbarer Speicherplatz) wird ausgewertet, der Wert `newvalue` wird zugewiesen.
  - `SETF`-Makro ermöglicht auch die Veränderung der Funktionsbindung und der property-list eines Symbols, sowie den Zugriff auf die Komponenten von Arrays und Listen.
  - Beispiel:

```
(setf var datum)
(setf (car var) datum) ; update 1. element
(setf (symbol-value 'var) datum)
```

SETF vs. SETQ

- SETQ erlaubt nur die Veränderung der ***Wertbindung*** von Symbolen.
- SETF kann auch ***Wert- und Funktionsbindung*** von Symbolen ändern und die ***Struktur von Listen modifizieren***.
  - Erstes Argument von SETF
    - Symbol oder
    - Funktionsaufruf (s. Bsp. letzte Folie), dessen erstes Element eine Selektorfunktion ist (z.B. CAR, CDR,CAAR, etc.)
  - Formen, die den Zugriff auf spezifische Komponenten von komplexen Objekten erlauben, werden als generalisierte Variablen bezeichnet.
    - Das durch den Funktionsaufruf selektierte Listenelement wird durch das Objekt ersetzt, zu dem der zweite Argumentbezeichner der SETF-Form evaluiert.

SETF vs. SETQ

- Die Mächtigkeit der Funktion SETF macht eine Reihe von Funktionen überflüssig, die in anderen LISP-Dialekten zur Modifikation von CONS-Strukturen verwendet werden:
  - z.B. die Funktionen RPLACA und RPLACD, die die Veränderung des CARs bzw. CDRs einer CONS-Struktur erlauben.

Globale Variablen

- `DEFPARAMETER Name Initialwert` – *Makro*
  - Initialwert muss angegeben sein.
  - Variable erhält den angegeben Wert.
  - Vorheriger Wert wird überschrieben.

- `DEFVAR Name [Initialwert]` – *Makro*
  - Sollte zur Definition von special-Variablen verwendet werden.
  - Der vorherige Wert einer globalen Variablen wird nicht überschrieben (schwer zu verstehen [Grund dynamische Bindung] , Vorsicht!).

- `DEFCONSTANT Name Initialwert` – *Makro*
  - Konstante mit Initialwert.

- Schreibweise globaler Variablen (mit ear muffs): *varname*

Bindung und Umgebung

- **Bindung** (binding)
  - Beziehung zwischen einer Variablen und dem ihr zugeordneten Wert.
- **Umgebung** (environment oder Geltungsbereich)
  - Menge aller zu einem gegebenen Zeitpunkt bestehenden Bindungen.

Variablentypen und Bindungsprinzipien

- Symbole können u.a. die Funktion von Variablen übernehmen.
- Wertzuweisung (Wertbindung) z.B. über SETF.
- Bei einem Funktionsaufruf werden die durch die Symbole der Lambda-Liste der Funktion bezeichneten Variablen (formale Parameter) an die Werte der Argumentbezeichner (aktuelle Parameter) gebunden.

Variablentypen und Bindungsprinzipien

- Unterschied zwischen diesen beiden Verwendungen von Symbolen:
  - im ersten Fall (außer bei Überdeckung): die Wertbindung hat **globale** Geltung (globale Variablen),
  - im zweiten Fall: die Wertbindung gilt nur innerhalb der Funktion (**lokale** Variablen).

➔ Geltungsbereiche von Variablen: Welche Bedingungen stellen es sicher/schließen es aus, dass ein Programmabschnitt auf eine zuvor generierte Variable zugreifen darf?

Geltungsbereichen von Variablen

- Der Geltungsbereich einer Variablen v ist der Bereich eines Programms, innerhalb dessen sie referenzierbar ist; d.h. innerhalb dessen die Evaluierung des sie bezeichnenden Symbols nicht zu einer Fehlermeldung der Form "unbound variable . . . " führt.
- Der Geltungsbereich von Variablen wird i.w. durch die Bindungsstrategie festgelegt.

Gebundene vs. Freie Variable

- Eine gebundene Variable bezüglich einer Funktion ist ein Symbol, das in der Parameterliste der Funktion enthalten ist.
- Eine freie Variable bezüglich einer Funktion ist ein Symbol, das nicht in der Parameterliste der Funktion enthalten ist.

- Die Werte freier Variablen sind lexikalisch (s.u.) festgelegt.

Lokale Variablen

- Variablen die in der Lambda-Liste einer Funktion generiert werden (aktuelle Parameter): lokale Variablen.

- Weitere lokale Variablen (z.B. zur Speicherung von Zwischenergebnissen) können im Funktionsrumpf deklariert werden (LET-Makro).

Wertzuweisungen / lokale Variablen: LET


```
LET ({var | ( var value )}*)}*
    {declaration}* {form}*
```
– *Special Form*

- Einrichtung neuer Variablenbindungen.
- `var` spezifiziert die innerhalb dieser Form gültigen **lokalen** Variablen.
- Wert von `var NIL` bzw. der angegebene Anfangswert.
- Diese Wertzuweisungen erfolgen parallel .
- Anschließend werden die Formen des Anweisungsblocks evaluiert.
- Wert der LET-Form = Wert der letzten Form.
- Geltungsbereich der in der LET-Form generierten Variablen beschränkt sich auf die LET-Form.
- Bei geschachtelten LET-Blöcken überdecken innere Variablen äußere Variablen gleichen Namens.

- `LET* ...` Wie `LET`, nur die Bindungen der Variablen erfolgen sequenziell.

Let – Beispiele


```
(defun anfang-&-ende (liste)
   (let ((anfang (first liste)) ; Berechne das erste Element der Liste.
         (ende (last liste)))   ; Berechne das letzte Element der Liste.
         (cons anfang ende)))
```

```
(defun spielerei1 (zahl)
   (let ((x zahl)
         (y (* zahl 2)))
         (* x y)))


(defun spielerei2 (zahl)
   (let ((x zahl)
         (y (* 2 x)))
         (* x y)))

(spielerei1 3) ; => 18
(spielerei2 3) ; => "Error: Variable X has no value"
```

Let – Beispiele


```
(defun spielerei3 (zahl)
   (let* ((x zahl)
          (y (* x 2)))
          (* x y)))

(spielerei3 3) ; => 18

(defun spielerei4 (zahl)
   (let ((x zahl)
         (y (* zahl 2))))
         (* x y))  ; x und y sind nur innerhalb von let bekannt

(spielerei4 3) ; => "Error: Unbound Variable X"
```

Let – Beispiele


```
(defun ueberdeckung ()
   (let ((a 3) (b 4))
      (let ((a 5) (c 6))  ; Das Äußere "a" wird durch
            (* a b c))))  ; das innere überdeckt.

(ueberdeckung) ; => 120
```

Bindungsstrategien

- Bisher: nur Funktionen mit gebundenen Variablen verwendet.
- Wie werden in Funktionen vorkommende freie Variablen gebunden?
- Unter welchen Bedingungen sind freie Variablen in Funktionen zulässig?
- In Common LISP gibt es 2 unterschiedliche Bindungsstrategien:
  - Lexikalische Bindung (wie in den meisten Programmiersprachen üblich),
  - Dynamische Bindung (für funktionale Programmiersprachen typisch).

Lexikalische Bindung

- Prinzip der lexikalischen Bindung ist der Standard.
- Geltungsbereich einer Variablen = Programmbereich (im Sinne von Textabschnitt, lexikalischer Abschnitt), innerhalb dessen sie definiert wurde.
  - D.h. der Geltungsbereich einer in einer Funktion generierten Variablen wird zum Zeitpunkt der Definition der Funktion und nicht zum Zeitpunkt ihrer Ausführung festgelegt.
- Geltungsbereich von Variablen, die in der Lambda-List einer Funktion generiert werden = Anweisungsblock, der den Funktionskörper bildet; der Versuch, von außerhalb dieses Bereiches auf diese Variablen zu referieren, führt zu Fehlermeldungen.
  - Damit wird auch verhindert, dass eine Funktion eine freie Variable enthalten kann, die durch eine sie aufrufende Funktion gebunden wird.

Dynamische Bindung

- War das ursprüngliche Bindungsprinzip in Lisp.
- Die formalen Parameter einer Funktion, nachdem sie beim Aufruf der Funktion gebunden wurden, bleiben so lange gebunden, bis die Evaluierung des Funktionsaufrufs mit der Rückgabe des Funktionswerts abgeschlossen wird.
- Dynamisch gebundene Variablen (auch Special-Variablen genannt) müssen explizit deklariert werden:

```
(declare (special var*))
```

  - Um innerhalb einer Funktion eine dynamisch gebundene Variable zu erzeugen, muss eine bereits generierte Variable als Special-Variable deklariert werden.

Dynamische Bindung: Beispiel

- Beispiel:

```
(defun uebergabe (argument)
   (declare (special argument))
   (rein-wie-raus))

(defun rein-wie-raus ()
   (print argument))
```

- Die Deklaration legt fest, dass für den Zeitraum der Ausführung von `uebergabe` die Variable `argument` dynamisch gebunden ist.
- Damit ist `argument` innerhalb des Aufrufs von `rein-wie-raus` referenzierbar.
- Die Aufrufkette legt fest, wo die Variable sichtbar ist.