`cond()` — Allgemeine Form

$$\texttt{(cond}\ \texttt{(}\langle\text{test}_1\rangle\ \langle\text{aktion}_{1,1}\rangle\ \ldots\ \langle\text{aktion}_{1,i}\rangle\texttt{)}$$
$$\vdots$$
$$\texttt{(}\langle\text{test}_n\rangle\ \langle\text{aktion}_{n,1}\rangle\ \ldots\ \langle\text{aktion}_{n,j}\rangle\texttt{))}$$

- die ‘$\texttt{(}\langle\text{test}_i\rangle\ \langle\text{aktion}_{i,1}\rangle\ \ldots\ \langle\text{aktion}_{i,j}\rangle\texttt{)}$’ Ausdrücke heißen *Klauseln*;
- Klauseln werden in der angegebenen Reihenfolge ausgewertet;
- für den ersten $\langle\text{test}_i\rangle$, der erfüllt ist, werden die Aktionen ausgeführt;
- der Wert des Gesamtausdrucks (`cond()`) ist der Wert der letzten Aktion;
- hat eine Klausel keine Aktion, gilt der Wert des Tests als Rückgabewert.

Verschachtelte (doppelte) Rekursion

- **Aufgabe:** Erweitere die Funktion `spegel()` (aka `reverse()`) so, daß auch eingebette Listen gespiegelt werden.

```
(defun spegel (list)
  (cond ((null list) nil)
        ((atom list) list)
        (t (append (spegel (rest list))
                   (list (spegel (first list)))))))
```

Restrekursion: Vergleich

nicht restrekursiv

```
(defun ! (i)
  (cond ((= i 0) 1)
        (t (* i (! (1- i))))))
```

- das Ergebnis des rekursiven Aufrufs geht in weitere Berechnung(en) ein;

restrekursiv

```
(defun ! (i)
  (help i 1))
(defun help (i n)
  (cond ((= i 0) n)
        (t (help (1- i)
                 (* i n)))))
```

- der rekursive Aufruf ist gleichzeitig der Rückgabewert der Funktion.