



## Outline

- 1 Einleitung
- 2 Einführung in C
- 3 C - Fehler finden und vermeiden
- 4 Fortgeschrittenes in C
- 5 C - Binärer Suchbaum
- 6 Einführung in Common Lisp
- 7 Lisp - Beispiele**
- 8 Einführung in Prolog
- 9 Formale Semantik



- Um Funktionen auf Listenelemente anzuwenden bietet Lisp viele Map-Funktionen
- `mapcar` wendet eine unäre Funktion sukzessive auf den Listkopf (`car`) an
- `(mapcar #'abs '(3 -4 2 -5 -6)) => (3 4 2 5 6)`
- `maplist` wendet eine unäre Funktion sukzessive auf den Rest der Liste (`cdr`) an
- `(maplist #'identity '(1 2 3)) => ((1 2 3) (2 3) (3))`
- Eine Verwandte methode ist `apply`
- `(apply #' + '(1 2 3)) => 6`
- `...#` signalisiert Interpreter, dass Symbol eine Funktion ist



## Lambdas

- Basis, sogenanntes Lambda-Kalkül
- Anonyme Funktionen am bekanntesten (C++, Python, Ruby,..)
- Lambda erzeugt aufrufbares Symbol
- Beispiel:  
`(lambda (x) (+ x 3))`
- Anwendung:  
`(mapcar (lambda (x) (if (evenp x) (print x)))  
'(1 2 3 4 5))`  
`=> 2, 4`



## Programmierbeispiel Binärbaum

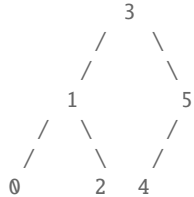
- Wie realisiert man einen Binärbaum in Lisp?



## Programmierbeispiel Binärbaum

- Wie realisiert man einen Binärbaum in Lisp?
- Als Liste von Listen von Listen...
- Beispiel:

```
1 ;  
2 ;  
3 ;  
4 ;  
5 ;  
6 ;  
7 ;  
8 (setf tree '(3 (1 (0 () ())) (2 () ())) (5 (4 () ())) ()))
```

A binary tree diagram with root node 3. Node 3 has left child 1 and right child 5. Node 1 has left child 0 and right child 2. Node 5 has left child 4. All other child positions are empty, represented by parentheses. The nodes are arranged in levels: level 0 (root) has 3; level 1 has 1 and 5; level 2 has 0, 2, and 4.



## Baumtraversierung

- Gegeben sei ein Baum `tree`. Definieren Sie eine Lispfunktion zum traversieren:
- In-Order
- Post-Order
- Pre-Order



## Einfügen

- Definieren Sie eine Lispfunktion zum Einfügen eines Integer Wertes in einen Binärensuchbaum



## Prüfen

- Entwerfen Sie ein Lisp Prädikat zum prüfen, ob ein Baum valide ist:
- Struktur stimmt?
- Ordnungsrelation stimmt?





## Fibonacci Zahlen rekursiv

- $f_n = f_{n-1} + f_{n-2}$  für  $n > 2$  und  $f_1 = f_2 = 1$



## Fibonacci Zahlen iterativ

- $f_n = f_{n-1} + f_{n-2}$  für  $n > 2$  und  $f_1 = f_2 = 1$



## Ackermann rekursiv

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$