# Funktionale Sprachen: LISP

- Garbage Collection:
  - Garbage: Speicher, der vom Programm aus nicht zugreifbar ist
  - sinnvoll: diesen freigeben und weiter nutzen

  - In Lisp: Speicher, auf den zugegriffen wird: Cons cells
  - → Garbage: Cons cells, die in weiterer Programmausführung nicht benötigt werden
  - Lisp erste Sprache mit automatischer Garbage Collection

  - Garbage Collection: Aufspüren von Garbage und Freigabe
  - Automatische Garbage Collection: „Free-Liste“ enthält freie Speicherplätze; wird Free-Liste knapp: „Programmpause“ während Garbage Collector arbeitet

# Funktionale Sprachen: LISP

- Verschiedenste Algorithmen zur Garbage Collection
- z.B. Mark-and-Sweep: jeder Speicherplatz hat tag (0: nicht referenziert, 1: referenziert); anfänglich alle tags 0; ausgehend von jedem direkt im Programm referenzierten Platz allen Links folgen und jeweils tag auf 1 setzen; am Ende alle 0-Plätze auf Free-Liste

- Nachteil autom. Garbage Collection: Laufzeitverlängerung
- Vorteil: zuverlässiger als Garbage Collection „von Hand“

- Bsp: (cdr (cons (cons ´A ´B) (cons ´C ´D)))

  aber: ((lambda (x) (car (cons x x))) ´(A B))

# Funktionale Sprachen: LISP

- Zusammenfassung Lisp:
  - elegante Sprache mit wenigen einfachen Ideen
  - entwickelt für symbolische Berechnungen
  - Innovation: Grund-Datenstruktur Liste (Cons cells),
    Grund-Kontrollstrukturen Rekursion (rekursive
    Funktionen) und *cond*itions
  - Lisp Syntax basiert/imitiert auf Lambda-Kalkül (äquivalent zu turing-vollständig, umfasst alle partiell rekursiven Funktionen)
  - erste Sprache mit automatischer Speicherorganisation (Garbage Collection)
  - Urvater verschiedener Lisp-Dialekte und anderer funktionaler Sprachen

# Funktionale Sprachen: LISP

- Besondere Eigenschaften anderer funktionaler Sprachen:
  - Haskell: lazy evaluation („Call-by-need“)
    - (Teil-)Ausdruck wird erst durch Wert ersetzt, wenn er benötigt wird
    - gilt auch für aktuelle Parameter
    - ermöglicht unendliche Strukturen:

```
intlist :: Integer -> [Integer]
intlist i = i : intlist(i+1)
```

```
zehnints = take 10 (intlist 0)   => [0,1,2,3,4,5,6,7,8,9]
von13bis21 = take 6 (intlist 13) => [13,14,15,16,17,18]
```

# Funktionale Sprachen: LISP

- Besondere Eigenschaften anderer funktionaler Sprachen:
  - ML: Typinferenz
    - automatische Typherleitung
    - über Regeln analog zur Typprüfung
    - dadurch teilweise Verzicht auf Typangaben
    - kollidiert mit autom. Typanpassung/Überladen

```
int a;
c = a + b;
```

```
fun  f (a, b, c)  =
           if b then a(b) else c+1;
```

# Funktionale Sprachen: LISP

- Besondere Eigenschaften anderer funktionaler Sprachen:
  - Haskell und ML: Currying
    - benannt nach Haskell Brooks Curry
    - Umwandlung einer Funktion mit mehreren Argumenten in Funktion mit nur einem Argument
    - erfunden von Schönfinkel/Frege

Statt

```
+: (Int, Int) → Int
```

in Haskell

```
+ :: Int → Int → Int
```