Programmieren in Common-Lisp
— Einführungskurs —

Jan Alexandersson

DFKI GmbH (Bau 43)
Raum 0.14
Tel +681 - 302 53 47
janal@dfki.de

Tilman Becker

DFKI GmbH (Bau 43)
Raum 0.21
Tel +681 - 302 52 71
becker@dfki.de

lisp-aufgaben@coli.uni-sb.de

Ziele dieses Kurses

- Alle TeilnehmerInnen erlernen die Grundlagen des Programmierens;
- sie beherrschen anschließend wichtige Techniken zum Schreiben von "schönen" (*strukturierten*) Programmen;
- sie kennen die grundlegenden Konzepte der Programmiersprache LISP;
- sie gewinnen praktische Erfahrung durch das Anwenden all dieser Kenntnisse
- haben dabei ihren Spaß und bestehen die Klausur ohne Probleme.

Ein Rezept

- (Möglichst) alle Vorlesungen besuchen; alle Übungen (erst) *selbst* bearbeiten;
- regelmäßig 2 – ? Stunden pro Woche *selbständig* (am Rechner) arbeiten;
- bei Fragen und Problemen: Kursleiter oder andere TeilnehmerInnen ansprechen.

Organisatorisches

Kür

- Vorlesung: Donnerstag 11:15 – 12:45 ;
- Übung: Mittwoch 11:15 – 13:00 ;
- Folienkopien im Handapparat der Bibliothek Computerlinguistik ;
- [Winston and Horn 1989] ausleihbar in der CoLi Lehrbuchsammlung;
- alle: '`lisp-kurs@coli.uni-sb.de`'.

Pflicht

- Übungsaufgaben; je 50 % der Punkte aus erster und zweiter Hälfte;
- Bearbeitungszeit: jeweils 1 Woche;
- individuelle (praktische) Rücksprache in einer Übung ungefähr in der Semestermitte;
- Semesterende: 90-minütige Klausur.

Aktuell im world-wide web: '`http://www.dfki.de/~becker/lispkurs/`'

WARUM

lernen wir überhaupt die uralte Programmiersprache Lisp???

Weil:

Syntax Die Synatax ist (sehr) einfach – wir können uns auf das Wesentliche konzentrieren.

Inkrementell Wir können sofort unsere Programme/Verschlimmbesserungen ausprobieren.

Mächtig CommonLisp enthält (fast) alle wesentlichen Programmierkonzepte – die gelernten Techniken sind direkt auf andere Programmiersprachen übertagbar.

und und und... :-)

Aber:

„Gute" Lispprogramme sind schwierig zu schreiben

Geschichte von LISP

1956	IPL	erstmals verwendet zur Strukturierung symbolischer Daten (Assembler-artige Sprache)
1958	LISP	**LIS**t **P**rocessor, entwickelt von John McCarthy als Aufsatz auf Fortran
1960	LISP1	Programmiersystem mit Interpreter, Compiler und Speicherverwaltung
1961	LISP1.5	Erster Lisp-Standard
19..	Maclisp, FranzLisp, Interlisp, TLC-Lisp, UCI-Lisp, NIL, Zeta-Lisp, Scheme	

1984 Common Lisp
1989 Common Lisp ANSI Standard (X3J13)

Anwendungsgebiete für LISP

- Formelmanipulation ($\rightarrow$ Macsyma)
- Modellbildung, Simulation ($\rightarrow$ Nintendo)
- Computergrafik ($\rightarrow$ Auto-CAD)
- Editoren ($\rightarrow$ Emacs)
- Entwicklungsumgebungen ($\rightarrow$ Genera)
- Systemprogrammierung ($\rightarrow$ Lisp Maschine)
- **Künstliche Intelligenz (KI):** Expertensysteme, Planungssysteme, Maschinelles Sehen, Automatisches Beweisen,Maschinelles Lernen
- **Computerlinguistik:** Morphologie, Syntaxanalyse (Parsing), Semantik, Sprach-Generierung, Dialogverabeitung
- **World Wide Web** Webserver (Viaweb/Yahoo, The K^2 System ...)

Literaturhinweise

- Patrick Henry Winston, Berthold Klaus Paul Horn: Lisp. Third Edition. Reading, Massachusetts (*Addison-Wesley*) 1989 (2. Ausgabe nicht akzeptabel).

- Guy L. Steele Jr.: Common-Lisp. The Language. Second Edition. Bedford, Massachusetts (*Digital Press*) 1990 (1. Ausgabe nicht akzeptabel). Auch im WWW auf der Lisp-Kurs Seite online! Manchmal gut zum Nachschlagen am Rechner.

- Gerald Gazdar, Chris Mellish: Natural Language Processing in LISP. Wokingham, England (*Addison-Wesley*) 1989.

- Eugene Charniak, Christopher K. Riesbeck, Drew V. McDermott, James R. Meehan: Artificial Intelligence Programming. Second Edition. Hillsdale, New Jersey (*Lawrence Erlbaum*) 1987.

Einige Grundbegriffe

Programmieren

Programmieren bedeutet, ein bestimmtes (Berechnungs-)Verfahren (einen *Algorithmus*) in einer speziellen (Computern verständlichen) Sprache (einer *Programmiersprache*) zu formulieren (*implementieren*).

Algorithmus

Ein Algorithmus ist eine (präzise formulierte) Verarbeitungsvorschrift zur Lösung eines Problems; Algorithmen für Computer heißen *Programme*. Interessante Algorithmen sind meist (i) endlich darstellbar, (ii) eindeutig und (iii) terminierend.

Beispiele für Algorithmen

Braten von Spiegeleiern

(1) Platte auf mittlere Hitze schalten;

(2) Pfanne auf Herdplatte stellen;

(3) Bratfett in die Pfanne geben;

(4) *Wenn* das Fett heiß genug ist,
dann Eier in die Pfanne schlagen;

(5) *Wenn* das Eiweiß fest ist,
dann Eier aus der Pfanne nehmen.

(6) Herd abschalten.

Multiplikation durch Addition

(1) x und y einlesen;

(2) $z := 0$;

(3) *Solange* $y \neq 0$

(4) $\quad y := y - 1$;

(5) $\quad z := z + x$;

(6) z ausgeben.

Entscheidend: Fähigkeit (Basisoperationen) des Prozessors

Elementare Konstrukte von Programmiersprachen

- **Datentypen** — Zahlen, Symbole, Zeichenketten (*strings*), Listen et al.;
- **Basisoperationen** — Addition, Zuweisung, Verkettung von *strings* et al.;
- **Kontrollstrukturen** — Wiederholung, Test, bedingte Ausführung et al.

Datentyp	**Common-Lisp Beispiele**
Zahlen	‘42’, ‘4711’, ‘3.1415’, ‘1/3’, ‘4.2E1’
Symbole	‘foo’, ‘bar’, ‘baz’, ‘pi’, ‘*foo*’, ‘common-lisp’
Zeichenketten	‘"a string"’, ‘"Common-Lisp"’, ‘"Grundkurs Lisp"’
Listen	‘(1 2 3)’, ‘(foo "bar" 42)’, ‘(+ 47 11)’, ‘()’

Syntax von Programmiersprachen

(Erweiterte) Backus-Naur Form

- Formale Grammatik zur Beschreibung der (Oberflächen-) Form (*Syntax*);
- erfaßt alle syntaktisch wohlgeformten Ausdrücke der Programmiersprache.

EBNF: Operatoren

- $\{\ x\ \}^*$ — Ausdruck x kann beliebig oft (auch gar nicht) auftreten;
- $\{\ x\ \}^+$ — Ausdruck x muß mindestens einmal auftreten (oder öfter);
- $[\ x\ ]$ — Ausdruck x ist optional (kann einmal oder gar nicht auftreten);
- $\{\ x \mid y\ \}$ — es kann entweder Ausdruck x oder Ausdruck y auftreten.

Common-Lisp Syntax — Beispiele

Beispiele

- $[\texttt{+}\,|\,\texttt{-}]\{\texttt{0}\,|\,\ldots\,|\,\texttt{9}\}^{+}$
- $\{[\texttt{+}\,|\,\texttt{-}]\{\texttt{0}\,|\,\ldots\,|\,\texttt{9}\}^{+}[\texttt{.}\{\texttt{0}\,|\,\ldots\,|\,\texttt{9}\}^{*}]\,|\,[\texttt{+}\,|\,\texttt{-}]\texttt{.}\{\texttt{0}\,|\,\ldots\,|\,\texttt{9}\}^{+}\}$

EBNF: syntaktische Variablen (Nicht-Terminale)

- $\langle digit\rangle ::= \{\texttt{0}\,|\,\texttt{1}\,|\,\texttt{2}\,|\,\texttt{3}\,|\,\texttt{4}\,|\,\texttt{5}\,|\,\texttt{6}\,|\,\texttt{7}\,|\,\texttt{8}\,|\,\texttt{9}\}$
- $\langle number\rangle ::= \{[\texttt{+}\,|\,\texttt{-}]\langle digit\rangle^{+}[\texttt{.}\langle digit\rangle^{*}]\,|\,[\texttt{+}\,|\,\texttt{-}]\texttt{.}\langle digit\rangle^{+}\,|\,\ldots\}$
- $\langle expression\rangle ::= \{\langle number\rangle\,|\,\langle symbol\rangle\,|\,\ldots\,|\,\langle list\rangle\}$
- $\langle list\rangle ::= \texttt{(}\,\langle expression\rangle^{*}\,\texttt{)}$

Listen und Numerische Ausdrücke

Listen

```
( 1 2 3 )   ( schrauben draehte knoepfe )  ( )
( 3 muttern knoepfe "zeichenkette" )
( ( schrauben muttern ) ( knoepfe "nadeln" ) draehte )
```

Falsch: `( ( schrauben ) nadeln ( )`

Numerische Ausdruecke

```
2  -3  0.5  +.732     .408E-9 1/3
( + 1 2.0 3 )  ( / 7 3 )
( max 34 78 4.0 )
```

Falsch: `( 1 + 2.0 3 )  ( 3 / 7 + 3 )`