Programmieren in Common-Lisp — Einführungskurs —

Jan Alexandersson	Stephan Oepen
DFKI GmbH (Bau 43)	Computerlinguistik (Bau 17)
Raum 0.14	Raum 3.02
☎ +681 - 302 53 47	☎ +681 - 302 41 76
janal@dfki.de	oe@coli.uni-sb.de

lisp-aufgaben@coli.uni-sb.de

Grundsätzliches

Ziele

- Alle TeilnehmerInnen erwerben wichtige Programmierkenntnis und -erfahrung,
- haben dabei ihren Spaß und bestehen Rücksprache und 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 9:15 – 10:45 h;
- Folienkopien im Handapparat der Bibliothek Computerlinguistik (2.11);
- [Winston and Horn 1989] ausleihbar in der CoLi Lehrbuchsammlung;
- alle: ‘`lisp-kurs@coli.uni-sb.de`’.

Pflicht

- 10 Übungsaufgaben; je 50 % der Punkte aus erster und zweiter Hälfte;
- Bearbeitungszeit: jeweils 1 Woche;
- individuelle (praktische) Rücksprache nach Übung 5 (Semestermitte);
- Semesterende: 90-minütige Klausur.

Aktuell im world-wide web: ‘`http://www.coli.uni-sb.de/~oe/lisp/`’

Literaturhinweise

- Patrick Henry Winston, Berthold Klaus Paul Horn: Lisp. Third Edition. Reading, Massachusetts (*Addison-Wesley*) 1989 (2. Ausgabe nicht akzeptabel).
- Apple Corporation: Macintosh Common Lisp. User and Reference Manuals. Cupertino, CA (*Apple Corporation*) 1992 (steht im CoLi Rechnerraum).
- Guy L. Steele Jr.: Common-Lisp. The Language. Second Edition. Bedford, Massachusetts (*Digital Press*) 1990 (1. Ausgabe nicht akzeptabel).
- 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 Spiegeleier

(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

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

EBNF: syntaktische Variablen (Nicht-Terminale)

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