Übungsblock 4

Lösungen 3. Übungsblatt

Aufgabe 1:

```
public static void main(String[] args){ //Syntaxanalyse; erzeugt div. Folgefehler
    if(args.length!=1) {
        System.err.println("Aufruf: java aufg3_1 <Datei oder Verzeichnis>“)
       //Syntaxanalyse; erzeugt Folgefehler
        …..
public static void delete(String datname) {  //Semant. Analyse; Fehler bei Aufruf
    …
    if (!f.exists()) fail("aufg3_1: Datei existiert nicht: " + datname);
      //Fehler in Programmlogik; fällt Compiler nicht auf
    ….
    if (f.isDirectory()) {                  //Syntaxanalyse
        String[] dateien = f.list();        //2 mal Syntaxanalyse


        …
```

Übungsblock 4

```
…
if (dateien.length > 0)
      //Fehler in Programmlogik; fällt Compiler nicht auf
      ….
boolean erfolg = f.delete();  //Semantische Analyse
    if(!erfolg) fail("aufg3_1: Loeschen fehlgeschlagen");  //mit Folgefehler
    hier
…
  protected static void fail(String s) throws IllegalArgumentException {
    //Semantische Analyse; mit Folgefehlern bei jedem Aufruf
}  // (letzte schließende Klammer fehlt) Syntaxanalyse
```

Das Programm erwartet genau einen Kommandozeilenparameter, sonst Fehlermeldung und Programmabbruch (6-9).
Dieser String wird als Parameter der Methode delete mitgegeben(11). In delete wird String als Datei-/Verzeichnisname angesehen und eine zugehörige File-Variable erzeugt.
Existiert Datei/Verzeichnis mit diesem Namen nicht (18), ist schreibgeschützt (19) oder ein Verzeichnis (20) und nicht leer (21-24), so wird in diesen Fehlerfällen die Funktion fail aufgerufen. Ansonsten wird die Datei/das Verzeichnis gelöscht (25). Schlägt das Löschen fehl, wird ebenfalls fail aufgerufen (26). Fail (29-31) selbst wirft eine IllegalArgumentException mit einer als Parameter übergebener Fehlermeldung.

Übungsblock 4

Lösungen 3. Übungsblatt

Aufgabe 2: Vor- und Nachteile eines hybriden Übersetzungssystems

+ viele Programmierfehler werden vor Interpretation erkannt
+ Zwischensprache einfacher/schneller zu interpretieren als Hochsprache
+ muss Programm oft ausgeführt werden: Quellcode wird nur einmal analysiert -> Programmausführung schneller

- starres Typsystem nötig/sinnvoll
- keine Code-Optimierung (wird erst auf Zwischensprache bzw. Maschinensprache gemacht)

Übungsblock 4

Lösungen 3. Übungsblatt

Aufgabe 3:

- ZAHL — Symbol
- 3-2-1-meins — falscher Ausdruck / Symbol (in clisp)
- 15,03 — falscher Ausdruck
- -25 — Zahl (Integer)
- FLOATP — Symbol
- STRING — Symbol
- DREI10 — Symbol
- 4+1 — falscher Ausdruck / Symbol (in clisp)
- 66/4 — Zahl (Rational)
- -33/11 — Zahl (Integer (clisp), Rational)

Übungsblock 4

Aufgabe 4:

• (+ 1 2 3 4 5 6 7 8 9)	45
• (+ -1 (- 3 1))	1
• (- (+ 3 5) (* 2 4) (/ 12 9))	-4/3
• (- (+ 3.0 5) (* 2 4) (/ 7 2))	-3.5

Aufgabe 5:

• (ATOM 5)	T
• (ODDP 5)	T
• (SYMBOLP 6)	NIL
• (EQUAL 3 3.0)	NIL
• (NOT (NOT T))	T
• (NUMBERP (SYMBOLP X))	NIL

Übungsblock 4

Schreibe Funktion *kopf-wechsel*, die zwei Listen als Parameter erhält und eine Liste erzeugt mit dem ersten Element der zweiten Liste als Kopf und dem Rest der ersten Liste (ohne Kopf) :

```
(defun kopf-wechsel (liste1 liste2)
      (cons (first liste2) (rest liste1)))
```

Übungsblock 4

Bestimme rekursiv die Anzahl Atome in einer Liste:

Idee: Anzahl Atome in Kopfelement + Anzahl Atome im Rest,

Kopf/Rest leer: 0, Kopf/Rest ein Atom: 1

```
(defun atome-in-liste  (liste)
   (cond  (  (null  liste)  0)   ; leere Liste?
             (  (atom liste)  1)  ; liste ist nur 1 Atom?
             (T  (+  (atome-in-liste (first liste))
                     (atome-in-liste (rest liste))))
   ))
```

Übungsblock 4

Übersicht über vordefinierte Funktionen:

Mathematische Funktionen:

+, -, *, /	(+ 3 4 2) → 9
<, >, <=, >=	(< z1 z2 z3) bedeutet: z1 < z2 < z3? → T/NIL
=, /=	(/= 5 3 5) → T
min, max	(max 5 9 2) →9
(exp x), (log x)	bzgl.e, (exp 1) → 2,71828…
(log x a)	log von x zur Basis a
(expt m n)	m^n
(sqrt x), (abs x)	
(signum x)	Vorzeichenfunktion, liefert 1 oder -1
(sin x), (cos x), (tan x), (asin x),…,(sinh x),…, (asinh x),..	
(floor x),(ceiling x)	Abrunden, Aufrunden (floor -1.1) → -2
(truncate x), (round x)	Ganzzahlanteil, Runden
(mod x y)	Rest von x:y

Übungsblock 4

Darstellung von Einzelzeichen (character): durch Voranstellen von #\
Darstellung von Strings: in doppelten Anführungszeichen
Sonderzeichen: ebenfalls Voranstellen von #\
Ausnahmen: doppelte Anführungszeichen selbst: \“
Backslash: \\ , Leerzeichen: #\Space
Zeilenwechsel: #\Newline, Tabulator: #\Tab
Beispiele: “ Erste Zeile #\Newline Zweite Zeile“
“Zeichen \“ und \\ und #\a“

Charakter-Funktionen:

char=, char<, char>, char <=,char>=	Vergleich zweier char (char< #\a #\b) → T
alpha-char-p, digit-char-p	Ist char Buchstabe? Ziffer? → T/NIL
lower-case-p, upper-case-p	Ist char Klein-/Großbuchstabe? → T/NIL
(char-int ch), (int-char z)	wandelt char int int und umgekehrt
(char-upcase ch), (char-downcase ch)	wandelt Klein- in Großbuchstabe und umgekehrt

Übungsblock 4

String-Funktionen:

(char string index) gib Zeichen aus String in der Stelle index zurück

string=, string<, string<=, string>, string>=, string/= Stringvergleich, beachtet Groß-/Kleinschreibung, liefert T/NIL

string-equal, string-less, string-greaterp, string-not-equal, string-not-lessp, string-not-greaterp Stringvergleich, ignoriert Groß-/Kleinschreibung, liefert T/NIL

(string-upcase string), (string-downcase string) wandelt String in Groß-/Kleinbuchstaben um

(string-trim char-seq string) entfernt am Stringanfang u. –ende sämtl. Zeichen, die in char-seq angegeben sind

(string-left-trim char-seq string) wie oben, aber nur am Stringanfang

(string-right-trim char-seq string) wie oben, aber nur am Stringende

zusätzlich alle Funktionen für Sequenzen anwendbar!

Übungsblock 4

Sequenzen: Listen, Vektoren, Strings

Funktionen für Sequenzen:

```
(elt  sequence idx)   ;liefert Element mit Index idx aus sequence
(subseq list start [end])        ;Teilsequenz von list ab Index start [bis
                                  ausschließlich end]
(copy-seq sequence)              ;erstellt Kopie
(length sequence)                ;Länge der Sequenz
(reverse sequence)               ;dreht Sequenz um
(remove item sequence)           ;entfernt alle Elemente item, gibt Kopie zurück
(remove-if pred sequence)        ;entfernt alle Elemente, die pred erfüllen
(delete item sequence)           ;löscht alle item-Elemente
(count item seq)                 ;zählt, wieviele item-Elemente in seq
(count-if pred seq)              ;zählt Elemente, die pred erfüllen
(find item seq)          ;gibt erstes Element, das mit item übereinstimmt,
                          zurück, sonst NIL
(find-if pred seq)       ;gibt erstes Element zurück, das pred erfüllt; sonst NIL
```

Übungsblock 4

```
(position item seq)          ;wie find, nur wird Position (Index) geliefert
(position-if pred seq)       ;wie find-if, nur wird Position geliefert
(search  seq1  seq2)         ;sucht seq1 als Teilsequenz von seq2, liefert
                             Anfangsindex oder NIL
(substitute new old seq)     ;ersetzt old durch new in seq
(sort seq pred)              ;sortiert seq nach einer durch pred gegebenen
                             Ordnungsrelation
```

Bei Funktionen mit Vergleichsbedingung (remove, position, count,…)
wird eql als Vergleich genommen;
andere Vergleichsoperation: mittels optionalem Parameter :test
angeben

```
(setf l1 ´(1 2 3 4) l2 ´(a b c d))
(setf li (list l1 l2))           ;li = ( (1 2 3 4) (a b c d))
(position ´(a b c d) li)         ;→NIL, da ungleiches Objekt
(position ´(a b c d) li  :test  #´equal)        ;→ 1 (Index!), da Test auf
   gleichen Inhalt
```

Übungsblock 4

Funktionen nur auf Teilsequenzen:
mittels optionaler Parameter :start und :end
(remove 4 ´(4 1 4 6 4 8 4 2) :start 2 :end 6) →
(4 1 6 8 4 2)

Beispiele Funktionen auf Sequenzen:
(position 3 ´(1 2 1 3 5)) → 3
(remove a ´(a b c d a b e)) →(b c d b e)
(remove 4 ´(3 9 5 3 2 1 5) :test #´<) → (3 3 2 1),
#´<: Funktionsaufruf
(remove-if #´evenp ´(1 2 3 4 5)) → (1 3 5)
(find-if #´evenp ´(1 3 6 2 7)) → 6
(substitute ´x ´a ´(a b a c)) → (x b x c)
(substitute ´x ´a ´(a b (a c))) → (x b (a c))
(sort ´(3 1 7 5 2) #´>) → (7 5 3 2 1)
(sort “ dbaacd“ #´char>=) → “ddcbaa“

Übungsblock 4

weitere Funktionen auf Listen:

```
(append list1 list2)        ;neue Liste aus list1, list2
(copy-list list)            ;Kopie von list, erfüllt equal, aber nicht eql
(push item list)            ;fügt item am Listenanfang ein
(pop list)                  ; gibt 1. Element, löscht es aus list
(endp list)                 ; T, wenn Listenende erreicht
(last list)                 ;liefert letztes Element (als Liste)
(butlast list)              ;liefert Liste bis auf das letzte Element
(subst new old list)        ;ersetzt old durch new, durchläuft rekursiv
                            Listenstruktur
(member item list)          ;kommt item in list vor, wird Teilliste ab item
                            zurückgegeben
(adjoin item list)          ;fügt item in list hinzu, falls noch nicht
                            vorhanden
(union l1 l2)               ;Vereinigungsmenge von l1 und l2
(intersection l1 l2)        ;Schnittmenge von l1, l2
(set-difference l1 l2)      ;Differenzmenge (in l1 und nicht in l2)
(subsetp l1 l2)             ;l1 Teilmenge von l2? → T, sonst NIL
```

Übungsblock 4

Was liefert folgender Ausdruck?

```
(APPEND (BUTLAST (CDR (LIST ‘A ‘B ‘(C D))))
                (REVERSE (CAR (CONS ‘(E F) ‘(LIST ‘G ‘H)))))
```

Lösung: (B F E)

Grund: (CDR (LIST ‘A ‘B ‘(C D))) liefert (B (C D)),

BUTLAST davon liefert (B),

(CONS ‘(E F) ‘(LIST ‘G ‘H)) liefert ((E F) LIST G H),

CAR davon liefert (E F),

REVERSE davon liefert also (F E)