

ITI0211- Loogiline programmeerimine

Loeng 3: Prologi andmestruktuurid

J.Vain

15.09.2020

# 4.1 Listid

- Listid esitavad *korteeže* ehk järjestatud elementide multihulki

```
[a, d, f, [s, f, []], d]
```

- List unifitseerub

- ühe muutujaga

```
List = [a, d, f, [s, f, []], d]
```

- listi erinevaid osi adresseerivate muutujatega, kui on mitte-tühi list

```
[Head|Tail]
```

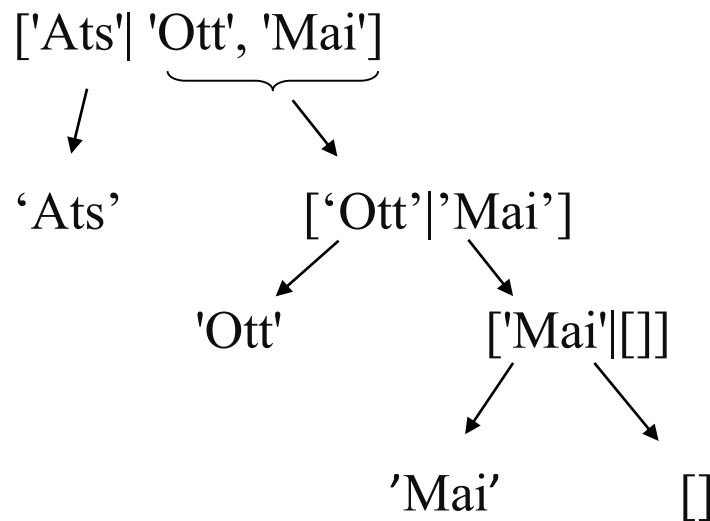
kus

- `Head` - listi pea, mis viitab otseselt listi esimestele elementidele
- `Tail` - listi saba, mis viitab listi kõigile ülejäänud elementidele
- `|` on eraldussümbol pea ja saba vahel.

# Listide päringu näiteid

- [ d, f, 4, 5, q]
- [ H | T ],
- [ \_ | T ],
- [ H | \_ ],
- [ E11, E12, E13 | Tail ]

# List on esitatav kahendpuuna



Otsing kahendpuul on väga kiire  
 $O(\log n)$

# Listi tüübikontroll

```
is_list(X) :-  
    var(X), !,  
    fail.  
is_list([]).  
is_list([_|T]) :-  
    is_list(T).
```

% Kui argument on väärtustamata muutuja, siis ta ei ole list  
% kui argument on tühelist  
% argument on mittetühi list kui tema saba on ka list

Märkus:

„ ...!, fail.“ reegli kehas suunab otsingu reeglist välja tulemusega `false`

# Näiteid listidest ja listipäringutest

```
assert(pere(['Ats', 'Mai', 'Ott'])).
```

```
?- pere(Liikmed).
```

```
Liikmed = ['Ats','Mai','Ott']
```

```
?- pere([Pea|Saba]).
```

```
Pea = 'Ats',
```

```
Saba = ['Mai','Ott']
```

**NB! Listi saba on alati list!**



```
?- pere([Isa,Emal|Lapsed]).
```

```
Isa = 'Ats'
```

```
Emal = 'Mai'
```

```
Lapsed = ['Ott']
```

# Listid ja rekursioon

- Listioperatsioonid on realiseeritud reeglina **rekursiivsete programmidega**
- **Pearekursioon** - tulemus leitakse rekursiivsete päringute ahelas päripidi liikudes:

```
element_of_list(↓A, ↓[ A | L]).  
element_of_list(↓A, ↓[ _ | L]) :-  
    element_of_list(↓A, ↓L).
```

- **Sabarekursioon** - tulemus leitakse rekursiivsete päringute ahelas tagurdamisel:

```
append(↓[], ↓A, ↑A).  
append(↓[A | B], ↓C, ↑[A | D]) :-  
    append(↓B, ↓C, ↑D).
```

# Listioperatsioonid: term – list teisendus “= ..”

## Näide 1:

```
?- ?isa(juku,peeter) =.. ?L.  
L = [isa, juku, peeter]
```

- Operaatori “= ..” rakendamisel termile on tulemuseks list, mille pea on predikaadi nimi ja saba predikaadi argumentid.

## Näide 2:

```
?- isa(Kes, Kellele) =.. L.  
Kes = _G492  
Kellele = _G493  
L = [isa, _G492, _G493]
```

- Operaatori “= ..” rakendamisel listile moodustub term, mille funktoriks on listi esimene element ja argumentideks ülejäänud elemendid.

## Näide 3:

```
?- [peep, ats, ott, mai] =.. L.  
L = ['.', peep, [ats, ott, mai]]
```



# Listioperatsioonid: konkatenatsioon (sabarekursioon)

`% append(?B, ?C, ?D)` parameetrid võivad olla nii sisendiks kui väljundiks

```
append([], A, A).
```

```
append([A|B], C, [A|D]):-
```

```
    append(B, C, D).
```

```
?- append (length([], 0).
```

```
[s,d,f], ↓[e,r,t], ↑X).
```

```
X = [s, d, f, e, r, t]
```

```
?- append(↓[s,d,f,g,h], ↑X, ↓[s,d,f,g,h,j,k,l]).
```

```
X = [j, k, l].
```

```
?- append(↑Q, ↑X, ↓[s,d,f,g,h,j,k,l]).
```

```
Q = [],
```

```
X = [s, d, f, g, h, j, k, l] ;
```

```
Q = [s],
```

```
X = [d, f, g, h, j, k, l] ;
```

```
Q = [s, d],
```

```
...
```

# Listioperatsioonid: elemendi eemaldamine listist (sabarekursioon)

% 1. argument on eemaldatav element;

% 2. argument on list, millest eemaldatakse elemendi kõik esinemised

% 3. argument on tagastatav list

```
remove(↓_, ↓[], ↑[]) .                                % Kui on tühelist
remove(↓S, ↓[S|T], ↑L) :-                             % Kui eemaldatav element on listi peas
    remove(↓S, ↓T, ↑L), !.
remove(↓S, ↓[U|T], ↑[U|L]) :-                         % Kui listi pea erineb eemaldatavast elemendist
    remove(↓S, ↓T, ↑L) .
```

```
?- remove( ats, [reet, peter, ott, ats], Uus_list) .
```

```
Uus_list = [reet, peter, ott]
```

# Listioperatsioonid: listi pikkuse leidmine (sabarekursioon)

```
?- length(↓List, ↑Length).
```

```
length([], 0).
```

```
length([S|T], M):-  
    length(T, N),  
    M is N + 1.
```

## Näide:

```
?- length([ e, r, t, w], A).
```

```
A = 4
```

# Listioperatsioonid: pöördlisti leidmine (sabarekursioon)

```
?- reverse(↓List, ↑R_list).
```

```
reverse([], []).
```

```
reverse([S|T], L):-  
    reverse(T, V),  
    append(V, [S], L).
```

## Näide:

```
?- reverse([1,2,3,4,5],R_list).
```

```
R_list = [5,4,3,2,1]
```

# Listioperatsioonid: elemendi sisalduvuse kontrollimine listis (pearekursioon)

```
?- member(↓Element, ↑List) .
```

```
member(S, [S|T]) .
```

```
member(S, [V|T]) :-
```

```
    member(S, T) .
```

**Näide:**

```
?- member([2,3,4], [c,s,[2,3,4],4,s]) .
```

```
true
```

# Listioperatsioonid: listi n-da elemendi leidmine (sabarekursioon)

```
?- nth_member(↑Element, ↓N, ↓List).
```

```
nth_member(S, 1, [S|_]).
```

```
nth_member(S, N, [_|L]):-
```

```
    T is N - 1,
```

```
    nth_member(S, T, L).
```

**Näide:**

```
?- nth_member(Element, 3, [w, 5, 6, g, h]).
```

```
S = 6
```

# Listioperatsioonid: permutatsioonid (pearekursioon)

```
?- permutation(↓List_1, ↓List_2).
```

```
permutation(Xs, Ys) :-                                % Kas list Xs on listi Ys elementide permutatsioon?
    insert(Xs, Sorted),
    insert(Ys, Sorted).
```

# Listioperatsioonid: leksikograafiline sorteerimine (sabarekursioon)

```
?- insert(↓List, ↑SortedList, ↓Ordering).
```

```
insert([], [], _).  
insert([X|L], Sorted_list, Ordering):-  
    insert(L, SL, Ordering),  
    insertx(↓X, ↓SL, ↑Sorted_list, ↓Ordering).
```

```
insertx(X, [], [X], _):-  
insertx(X, [A|L], [A|M], Ordering):-           % Asetab X väärtuse sorteeritud listi [A|L] (2. parameeter)  
    P =.. [Ordering, A, X], call(P),           % Kui A < X  
    insertx(X, L, M, Ordering).  
insertx(X, [A|L], [X,A|L], _):-                 % Kui X < A  
    A>=X.
```

Sorteerimispredikaadi defineerimine:

```
Ordering:= '<'           – aritmeetiline järjestamine;  
Ordering:= 'aless'      – leksikograafiline järjestamine
```



# Listioperatsioonid: leksikograafiline soreerimine (järg)

```
alless (X, Y) :-          % aatomite leksikograafilise järjestuse kontrollimine  
name (X, L), name (Y, M), allessx (L, M) .
```

```
allessx ([], [_|_]) .  
allessx ([X|_], [Y|_]) :- X < Y.  
allessx ([H|Q], [H|S]) :- allessx (Q, S) .
```

- **Märkus:** `name (Atom, List)` teisendab aatomi sümbolite koodide listiks