

Logikai programozás

4.

ISMÉTLÉS – n!

fakt(N,F) :- fakt(N,1,1,F).

fakt(N,I,P,F) :- I=<N, UjP is P*I, Ujl is I+1,
fakt(N,Ujl,UjP,F).

fakt(_,_ ,F,F).

Mi a fakt/4 utolsó két paraméterének szerepe?

Az egyik kezdőérték, a másik végérték.

Pontosabban: ld. köv. dia

ISMÉTLÉS

Akkumulátor (gyűjtőargumentum-pár):

Ugyanahhoz a mennyiséghez tartozó változópár:

- az egyik a mennyiség belépéskori értéke
- a másik a kilépéskor érvényes érték.

Pl.: ha a hívás: `hossza(Lista, 0)`

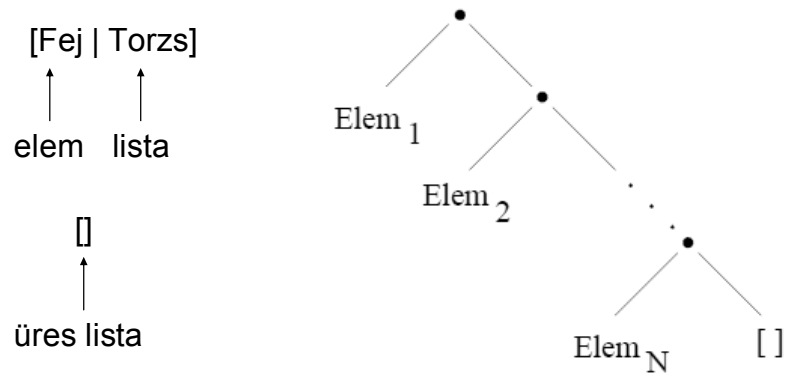
akkor honnan tudjuk meg a kérdéses hosszat?

Helyes megoldás: `hossza(Lista, 0, Eredmeny)`

Vagy: `hossza(Lista, H) :- hossza(Lista, 0, H).`

ISMÉTLÉS – LISTÁK

Lista: elemek felsorolása



ISMÉTLÉS – LISTÁK

Listafeldolgozás:

?- feldolgoz(Lista).

feldolgoz([Fej | Torzs]) :- muvelet(Fej), feldolgoz(Torzs).
+ leállási feltétel

De lehet:

feldolgoz([Fej | Torzs]) :- feldolgoz(Torzs), muvelet(Fej).

Vagyis a tényleges művelet lehet:

a/ a listába „befelé” menet

b/ „kifelé” jövet.

LISTÁK – PÉLDÁK

Listaelemek olvasása:

olvas([Fej|Torzs]) :- read(Fej), olvas(Torzs).
olvas([]).

Probléma: Sohasem áll meg. ☹

olvas([]).
olvas([Fej|Torzs]) :- read(Fej), olvas(Torzs).

Probléma: Azonnal megáll, eredmény: []

LISTÁK – PÉLDÁK

Listaelemek olvasása:

```
olvas([Fej|Torzs]) :- read(Fej),  
                    Fej \= vege,  
                    olvas(Torzs).  
olvas([]).
```

FONTOS:

- a felhasználóval mindig közölni kell, hogy mi a végjel
- és azt is, hogy adatot várunk vagyis:

```
olvas([Fej|Torzs]) :- writef(...), read(Fej),...
```

„Odafelé” vagy „visszafelé” tölti fel a listát?

LISTÁK – PÉLDÁK

Listaelemek olvasása másképp:

```
olvas(L) :- olvas([],L).
```

```
olvas(Eddig, Lista) :- writef(...), read(Elem),  
                      Elem \= vege,  
                      olvas([Elem|Eddig], Lista).
```

```
olvas(Lista,Lista).
```

„Odafelé” vagy „visszafelé” tölti fel a listát?

Eredmény: a beolvasási sorrend fordítottja.

LISTÁK – PÉLDÁK

Listaelem törlése:

```
% torol( elem, régi_lista, új_lista)
```

```
torol(Elem, [Elem | Torzs], Torzs).
```

```
torol(Elem, [Fej | Torzs], [Fej | Torzs2] ) :-  
    torol(Elem, Torzs, Torzs2).
```

```
torol( _, [], [] ).
```

Többszörös előfordulás törlése:

```
torol(Elem, [Elem | Torzs], Torzs2) :-  
    torol(Elem, Torzs, Torzs2).
```

LISTÁK – PÉLDÁK

```
torol(Elem, [Fej | Torzs], [Fej | Torzs2] ) :-  
    torol(Elem, Torzs, Torzs2).
```

```
e    a, b, e, c, e, d, f    a, b, c, d, f
```

```
e    b, e, c, e, d, f    b, c, d, f
```

```
torol(Elem, [Elem | Torzs], Torzs2) :-  
    torol(Elem, Torzs, Torzs2).
```

```
e    e, b, e, c, e, d, f    b, c, d, f
```

```
e    b, e, c, e, d, f    b, c, d, f
```

LISTÁK – PÉLDÁK

Listaelem beszúrása:

```
% torol( elem, régi_lista, új_lista)
```

```
torol(Elem, [Elem | Torzs], Torzs).
```

```
torol(Elem, [Fej | Torzs], [Fej | Torzs2] ) :-  
                                         torol(Elem, Torzs, Torzs2).
```

```
torol( _, [], [] ).
```

Vagyis ugyanez, csak ha a hívás pl.

```
torol( a, [b,a,d,e], L ) – akkor törlés
```

```
torol(a, L, [b,d,e]) – akkor beszúrás  
                    (több alternatív megoldás)
```

LISTÁK – PÉLDÁK

Listák összefűzése:

```
% fuz(egyik, masik, harmadik)
```

```
fuz([Fej|L1Torzs], L2, [Fej|L3Torzs] ) :-  
                                         fuz(L1Torzs, L2, L3Torzs).
```

```
fuz([], L, L).
```

```
a, b, c      d, e      a, b, c, d, e
```

```
  b, c      d, e      b, c, d, e
```

LISTÁK – PÉLDÁK

fuz([Fej|L1Torzs], L2, [Fej|L3Torzs]) :-
fuz(L1Torzs, L2, L3Torzs).
fuz([], L, L).

[trace] 2 ?- fuz([a,b,c],[d,e],L).
Call: (6) fuz([a, b, c], [d, e], _G549) ? creep
Call: (7) fuz([b, c], [d, e], _G631) ? creep
Call: (8) fuz([c], [d, e], _G634) ? creep
Call: (9) fuz([], [d, e], _G637) ? creep
Exit: (9) fuz([], [d, e], [d, e]) ? creep
Exit: (8) fuz([c], [d, e], [c, d, e]) ? creep
Exit: (7) fuz([b, c], [d, e], [b, c, d, e]) ? creep
Exit: (6) fuz([a, b, c], [d, e], [a, b, c, d, e]) ?
L = [a, b, c, d, e].

LISTÁK – PÉLDÁK

fuz([Fej|L1Torzs], L2, [Fej|L3Torzs]) :-
fuz(L1Torzs, L2, L3Torzs).
fuz([], L, L).

[trace] 3 ?- fuz([a,b,c],L,[a,b,c,d,e]).
Call: (6) fuz([a, b, c], _G593, [a, b, c, d, e]) ? creep
Call: (7) fuz([b, c], _G593, [b, c, d, e]) ? creep
Call: (8) fuz([c], _G593, [c, d, e]) ? creep
Call: (9) fuz([], _G593, [d, e]) ? creep
Exit: (9) fuz([], [d, e], [d, e]) ? creep
Exit: (8) fuz([c], [d, e], [c, d, e]) ? creep
Exit: (7) fuz([b, c], [d, e], [b, c, d, e]) ? creep
Exit: (6) fuz([a, b, c], [d, e], [a, b, c, d, e]) ? creep
L = [d, e].

LISTÁK – PÉLDÁK

Lista megfordítása:

```
% fordit(regi_lista, új_lista)
```

```
fordit([],[]).
```

```
fordit([F|T],Forditott):-fordit(T,Tford), fuz(Tford,[F],Forditott).
```

Logikailag jó, de rossz hatásfokú! (n^2 nagyságrendű)

LISTÁK – PÉLDÁK

Lista megfordítása (hatékonyabb):

```
% fordit(regi,forditott)
```

```
% fordit(regi, temp, forditott)
```

```
fordit(Regi, Forditott) :- fordit(Regi, [], Forditott).
```

```
fordit([F|T], Eddig, Forditott) :- fordit(T, [F|Eddig], Forditott).
```

```
fordit([], L,L).
```

a,b, c	[]	c, b, a
b, c	a	c, b, a
c	b, a	c, b, a
[]	c, b, a	c, b, a

LISTÁK – PÉLDÁK

Lista elemeinek szétválogatása:

```
% valogat(eredeti_lista, jo_lista, nemjo_lista)
```

```
% jó: a feltételnek megfelelő
```

```
valogat([],[],[]).
```

```
valogat([F|T],[F|Jo], Rossz) :- feltetel(F),  
                                     valogat(T, Jo, Rossz).
```

```
valogat([F|T], Jo, [F|Rossz]) :- valogat(T, Jo, Rossz).
```

Mikor válogat? A listába „befelé” menet, vagy „kifelé”?

A másik irány: HF

LISTÁK – PÉLDÁK

Ezek után:

Hogyan lehet összeadni egy olyan listában lévő számokat, amelyek vegyesen tartalmazhat számokat is és nem számokat is?