Prolog 2.

Készítette: Szabó Éva

Relációs operátorok

=<, <, >, >=

Kisebb-egyenlő (!), a többi ugyanaz, mint máshol.

== - egyenlő

\= - nem egyenlő

pl.:

```
nagyobb(A,B,A) :- A > B.
nagyobb(A,B,B) :- A =< B.
```

Összetett term – A lista

- **[ ]** - k között felsorolt **azonos** típusú elemek rendezetlen halmaza.
- Az elemeket **,**-vel választjuk el.

- pl.: [1,2,3,4,5]
-
- Head-tail felbontás: [H | T]

A prolog a karaktereket az ASCII-kódjukkal azonosítja, tehát minden string valójában egész számokból álló listának felel meg.

pl.:

B = ”abc”.

Ennek kimenete: *B = [97, 98, 99]*

Listák feldolgozása

1. Határozzuk meg egy lista első elemét!

A feladat elvégzéséhez definiálunk egy olyan tényt, ami akkor igaz, ha az első paraméterben lévő lista első eleme illeszkedik a második paraméterben megadott elemmel. Ha az első paraméterben megadunk egy listát, a második paraméter pedig egy szabad változó, akkor illeszkedni fognak, és a szabad változó értéke lesz az első elem értéke.

elso([H|_],H).

pl.: *elso([1,2,3],X).* → X = 1

2. Határozzuk meg egy lista utolsó elemét!

Érezzük, hogy itt már rekurziót kell alkalmaznunk. Ha rekurziót használunk, akkor kell bázis eset. Jelen esetben az lesz, amikor egyetlen elemű a lista, mert akkor az első elem egyben az utolsó is. Továbbá ha nem egy elemű a lista, akkor tudjuk hogy az egész lista utolsó eleme megegyezik a farok utolsó elemével.

utolso([H],H). utolso([_|T], X) :- utolso (T,X).

3. Határozzuk meg a lista elemszámát!

Haskellben már megírtuk, a megoldás teljesen hasonló módon történik. Tudjuk, hogy az üres lista elemszáma 0, ez tény. Egyébként a lista elemszáma a farok elemszámánál 1-gyel több. A számláló növeléséhez használjuk az ***is*** operátort!

len([],0).

len([H|T],N) :- len(T,N1), N is N1 + 1.

4. Összegezzük egy egész számokból álló lista elemeit!

Az alapeset a szokásos, az üres lista elemeinek összege 0. Általános esetben pedig a lista elemeinek összege a farok összege + a fej. Ez adja a következő rekurziót:

```
sum([],0).
sum([H|T],N) :- sum(T,N1), N is N1 + H.
```

5. Fűzzünk össze két listát, azaz az első lista utolsó eleme után jöjjenek a második lista elemei!

Az alapeset jelen esetben legyen az, amikor az első lista üres lista, ekkor az összefűzés után a második listát kapjuk. Általános esetben pedig tudjuk, hogy az első lista első eleme az összefűzött listának is első eleme lesz, tehát elég összefűzni az első lista farkát a második listával, és elé tenni az első lista első elemét a következőképpen:

append([],L,L).

append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

6. Fordítsuk meg egy lista elemeit!

Pontosan ugyanazzal a logikával dolgozunk, mint Haskellben. Megfordítjuk a farkát, és a végére tesszük a fejet. Ehhez szükségünk lesz az előző feladatban definiált append predikátumra.

rev([],[]).

rev([H|T],L) :- rev (T,L1), append(L1,[H],L).

7. Adott egy lista és egy elem. Döntsük el, hogy az elem benne van-e a listában!

Egyszerűen ha az elem a lista első eleme, akkor készen vagyunk. Ha nem, akkor rekurzívan megnézzük, hogy a farok résznek eleme-e.

member([E|_],E).

member([_|T],X) :- member(T,X).

8. Adott egy lista és egy elem. Töröljük ki az elemet a listából!

A módszer itt is hasonló, mint Haskellben volt. Ha az első elemet kell törölni, akkor szimplán elhagyjuk, egyébként meghagyjuk.

```
del([H|T],H,T).
del([H|T],X,[H|L]) :- del(T,X,L).
```