Deklaratív Programozás 6. gyakorlat
Prolog programozás, ZH mintafeladatok

"univ" azaz =.. /2 beépített eljárás
==========

```
    Hívási módok:
        +Kif =.. ?Lista
        -Kif =.. +Lista
    Jelentése: igaz, ha
    o Kif = Str(A1,...,An) és Lista = [Str,A1,...,An],
      ahol Str egy névkonstans és A1,...,An tetszőleges kifejezések; vagy
    o Kif = C, és Lista = [C], ahol C egy (név- vagy szám-)konstans.
```

Mintapélda: polinom behelyettesítési értékének kiszámítása (222. fólia)

Nevezzük formulának az 'x' névkonstansból és számokból a '+' és '*' operátorokkal felépülő kifejezéseket.
A feladat: Egy formula értékének kiszámolása egy adott x érték esetén.

```
% erteke(Kif, X, E): A Kif formula x=X helyen vett értéke E.
erteke(x, X, E) :-
    E = X.
erteke(Kif, _, E) :-
    number(Kif), E = Kif.
erteke(K1+K2, X, E) :-
    erteke(K1, X, E1), erteke(K2, X, E2), E is E1+E2.
erteke(K1*K2, X, E) :-
    erteke(K1, X, E1), erteke(K2, X, E2), E is E1*E2.

| ?- erteke((x+1)*x+x+2*(x+x+3), 2, E).     ==>   E = 22 ? ; no
```

Az erteke/3 predikátum két utolsó, rekurzív klóza teljesen analóg szerkezetű.
Az =.. /2 beépített eljárás (ejtsd univ) segítségével ezek összevonhatók:

```
erteke(Kif, X, E) :-
    Kif =.. [Op,K1,K2], erteke(K1, X, E1), erteke(K2, X, E2),
    EKif =.. [Op,E1,E2], E is EKif.
```

Kövessük végig az erteke((x+1)*x, 2, E) hívás lefutását, feltételezve, hogy a rekurzív hívások helyes eredményt adnak!

Vegyük észre, hogy az EKif változó az =.. végrehajtásakor a 3*2 (struktúrakifejezés) értéket veszi fel, amit az E is Ekif kiértékel, és előáll az E = 6 végeredmény!

Vegyük észre, hogy az új változat nemcsak a + és * műveletekre, hanem minden az is/2 által elfogadott bináris műveletre működik!

```
| ?- erteke(exp(100,min(x,1/x)), 2, V). ==> V = 10.0 ? ; no
```

További hasznos eljárások:
==========

atom_codes/2: atomok szétszedése és összerakása

```
   Hívási módok:
       atom_codes(+Atom, ?Codes)          Atom - tetszőleges névkonstans
       atom_codes(-Atom, +Codes)          Codes - karakterkódok listája.
   Jelentése, az Atom névkonstanst alkotó karakterek listája Codes.

        Példák:
        | ?- atom_codes(a0b, L).             ===>      L = [97,48,98] ? ; no
        | ?- atom_codes(A, [98,48,97]).      ===>      A = b0a ? ; no
```

prefix_length/3 eljárás (SICStus lists könyvtár, SWI-ben sajnos nincs)

```
   Hívási módok:
       prefix_length(?List, ?Prefix, ?Length)
   Jelentése: Prefix a List lista kezdőszelete, és Prefix hossza Length.
   Véges futású, ha Length adott, vagy a List és Prefix listák közül legalább
   az egyik zárt végű.
```

Lásd még 3. gyakorlat +7 számú szorgalmi feladat.

Meta-logikai beépített eljárásokkal kapcsolatos feladatok
==========

1. Atomok szeletelése

Egy A atom prefixumának nevezünk egy P atomot, ha P az A első valahány karakterét tartalmazza, az A-beli sorrend megtartásával.

```
% atom_prefix(+Atom, ?Prefix, +N): Atom-nak Prefix N hosszú prefixuma.
% Másszóval: az Atom első N karakteréből képzett névkonstans a Prefix atom.

| ?- atom_prefix(abcde, Prefix, 0).
Prefix = '' ? ; no
| ?- atom_prefix(abcde, Prefix, 3).
Prefix = abc ? ; no
| ?- atom_prefix(abcde, Prefix, 5).
Prefix = abcde ? ; no
| ?- atom_prefix(abcde, Prefix, 6).
no
```

Nem használhatja a sub_atom/5 beépített eljárást!
Ötlet: használja az atom_codes és prefix_length fent ismertetett eljárásokat.

2. Egy általános Prolog kifejezés részkifejezéseinek vizsgálata

```
% mern(+K, +N): A K általános Prolog kifejezésben előforduló összes egész
% szám határozottan nagyobb mint N (mern = minden egész részkifejezése
% nagyobb mint)

| ?- mern(1, 1).                          ==> no
| ?- mern(1, 0).                          ==> yes
| ?- mern(0.0, 1).                        ==> yes
| ?- mern(f(X,[1,3,b],g(2,1,3)), 0).      ==> yes
| ?- mern(f(X,[1,3,b],g(2,1,3)), 1).      ==> no
```

Megjegyzések:

a. A "K1 Prolog kifejezésben előfordul a K2 kifejezés" relációt reflexívnek tekintjük, azaz egy K kifejezésben önmaga mindenképpen előfordul. Ez a megjegyzés vonatkozik az ezután következő feladatokra is.
b. Vigyázzon arra, hogy a kifejezésben változók is előfordulhatnak.

3. Általános Prolog kifejezés bizonyos részkifejezéseinek felsorolása

```
% reszatom(+K, ?A): A a K általános Prolog kifejezésben előforduló atom.

| ?- reszatom(a, X).
X = a ? ; no
| ?- reszatom(f(X,[1,3,b],g(2,1,a0)), A).
A = b ? ;
A = [] ? ;
A = a0 ? ; no
```

Megjegyzés: a struktúranevet nem tekintjük a struktúrakifejezés részatomjának.

Segítségként magyar nyelven megfogalmazunk egy kijelentést, amely Prologba átírható:

Az "A egy a K kifejezésben előforduló atom" állítás két esetben állhat fenn
1. K maga egy atom -- ilyenkor A = K
2. K összetett, argumentumlistája KL, és KL-nek van olyan K1 eleme, hogy "A1 egy a K1 kifejezésben előforduló atom" teljesül -- ilyenkor A = A1

4. Általános Prolog kifejezés bizonyos részkifejezéseinek akkumulálása

```
% osszege(+K, ?Ossz): Ossz a K kifejezésben előforduló egész számok
% összege.

| ?- osszege(a, S).
S = 0 ? ; no
| ?- osszege(1, S).
S = 1 ? ; no
| ?- osszege(f(X,[1,3,b],g(2,1,a0)), S).
S = 7 ? ; no
```

A következő feladat példa arra, hogy a Deklaratív Programozás NZH Prolog részében milyen "mitírki" jellegű feladatokat kell megoldani.

---

5. "Mitírki" jellegű NZH mintafeladat

==========

Tekintse az alábbi Prolog programot és döntse el mindegyik célról, hogy hogyan fut le:

- sikerül,
- meghiúsul, vagy
- hibát jelez!

Sikeres futás esetén adja meg az összes olyan változó értékét, amelynek neve nem aláhúzás-jellel (_) kezdődik. Ha egy cél többféleképpen is sikerülhet, akkor adja meg az összes lehetséges behelyettesítést pontosvesszőkkel elválasztva!

Mindegyik célt a Prolog interpreternek önmagában adjuk oda, azaz futásának kezdetén a célban előforduló változóknak nincs értéke. Feltételezzük, hogy a ‘lists’ könyvtár be van töltve.

```
p(1).
p(2).
p(X) :- X > 1.

m(2, 0).
m(1, 2).
m(1, 3).
m(2, 1).
m(_, 4).

q(X) :- m(_, X), p(X).
```

(a) ?- select(1, [2,X,3], L).
(b) ?- atom_codes(abc, [_|L]), atom_codes(X, L).
(c) ?- \+ \+ X = 1, X = 2.
(d) ?- m(1, X).
(e) ?- q(X).

A következő feladatok bemutatják, hogy a NZH Prolog részében milyen jellegű programozási feladatokat kell megoldani, milyen szerkezetben.
Két, egymásra épűlő feladatot kell kidolgozni: először egy segédeljárást kell elkészíteni, majd ezt felhasználva egy, a teljes feladat megoldását előállító predikátumot kell megírni. Az is megengedett, hogy csak a teljes feladatot oldja meg, akár saját maga választotta segédeljárással (amelyhez feltétlenül írjon fejkommentet), vagy akár segédeljárás nélkül.

Prolog programozási NZH mintafeladat

==========

6. Segédeljárás:

Írjon Prolog nyelven egy olyan eljárást, amely előállítja egy konstans értékét egy helyettesítési lista alapján. A helyettesítési lista minden eleme Név-Szám alakú, ahol Szám a Név névkonstans helyettesítési értéke. Egy számkonstans helyettesítési értéke önmaga, egy a helyettesítési listában nem szereplő atom helyettesítési érteke pedig 0. Ha egy névkonstans többször szerepel a helyettesítési listában, akkor az első előfordulást szabad csak figyelembe venni.

```
% helyettesitese(K, HL, E): A K konstansnak a HL behelyettesítési
% lista szerinti értéke E.
% :- type helyettesites ---> atom - number.
% helyettesitese(univ::in, list(helyettesites)::in, number::out).
```

Példák:

```
| ?-  helyettesitese(y, [x-1,y-2,z-3], H).
H = 2 ? ; no
| ?-  helyettesitese(u, [x-1,y-2,z-3], H).
H = 0 ? ; no
| ?-  helyettesitese(x, [x-1,z-3,x-2], H).
H = 1 ? ; no
| ?-  helyettesitese(4, [x-1,y-2,z-3], H).
H = 4 ? ; no
```

7. Teljes feladat:

A helyettesitese/3 eljárás segítségével (vagy anélkül) írjon olyan Prolog eljárást, amely egy többváltozós kifejezés adott lista szerinti behelyettesítési értékét számítja ki! A kifejezést egy olyan Prolog adatstruktúrával adjuk meg, amely atomokból és számokból az ’is’ beépített eljárás által megengedett egy- ill. kétargumentumú műveletekkel épül fel.

```
% erteke(Kif, Hely, Ert): A Kif kifejezés értéke a Hely behelyettesítési
% lista által adott helyen Ert.
% :- pred erteke(univ::in, list(helyettesites)::in, number::out).
```

Példák:

```
| ?- erteke(x, [x-22], E).
E = 22 ? ; no
| ?- erteke(-x, [x-5], E).
E = -5 ? ; no
| ?- erteke(33, [x-22], E).
E = 33 ? ; no
| ?- erteke((x+y)*(x+y)+1, [x-1,y-2], E).
E = 10 ? ; no
| ?- erteke(x*abs(z)+x, [x-1,z-(-2)], E).
E = 3 ? ; no
| ?- erteke(asin(x), [x-0.5], E1), erteke((2*cos(x))**2, [x-E1], E).
E = 2.9999999999999996, E1 = 0.5235987755982989 ? ; no
```