Deklaratív Programozás gyakorlat
Prolog programozás 1.

"V" Témakör: Prolog végrehajtás

Az alábbi V1-V4 feladatokhoz definiáljuk egy személy felmenőjét a következő két állítással:

X felmenője Y, ha Y X egyik szülőjének felmenője.
X felmenője X.

Tekintse a fenti definíciót megvalósító és a "szülője" kapcsolatot is leíró alábbi Prolog programot.

```
% f(X, Y): X felmenője Y.
f(X, Y) :- sz(X, S), f(S, Y).     % (f1)
f(X, X).                          % (f2)

% sz(X, Y): X szülője Y.
sz(a, b).                         % (sz1)
sz(a, c).                         % (sz2)
sz(c, d).                         % (sz3)
```

V1. Rajzolja fel a

```
                     | ?- f(a, V).                                   (1)
```

kérdéshez tartozó keresési fát! A keresési fa alapján írja le, hogy a Prolog rendszer erre a kérdésre milyen válaszokat ad, és milyen sorrendben adja ezeket!

V2. (szorgalmi, otthoni feladat) Rajzolja fel a | ?- f(U, V). kérdéshez tartozó keresési fát; ennek alapján írja le, hogy a Prolog rendszer erre a kérdésre milyen válaszokat ad, és milyen sorrendben adja ezeket!

V3. Adja meg, hogy az (1) kérdésre milyen sorrendben kapjuk meg a válaszokat, ha a fenti programban az (f1) és (f2) klózok sorrendjét megcseréljük!

V4. (szorgalmi, otthoni feladat) Mi történik az (1) kérdés futtatásakor, ha az (f1) és (f2) klózok ebben a sorrendben követik egymást, de az (f1) klóz törzsében levő két hívás sorrendjét megcseréljük?

Megoldását ellenőrizheti a Tau Prolog segítségével, ld. http://tau-prolog.org/sandbox/, a Derivation tree visualization fül kiválasztásával.

"P". Témakör: programok írása

Megoldásait ellenőrizheti az ETS segítségével: https://dp.iit.bme.hu/ets/, Gyakorlás menüpont (bal oldalon fent), utolsó témakör (Prolog programozás), a lenyíló menüben Prolog 1. gyakorlat, ill. Prolog 1. gyakorlat szorgalmi feladatok menüpont választásával.

A programozási feladatok megoldásaként olyan Prolog eljárást kell írnia, amely megfelel az adott fejkommentnek. Ha külön nem kérjük, akkor nem szükséges, hogy jobbrekurzív (iteratív) eljárásokat írjon, de természetesen a jobbrekurzív változatot ilyenkor is elfogadjuk.

Bármely feladat megoldásához felhasználhat korábbi sorszámú feladatokban definiált eljárásokat. Ha ilyeneken kívül is szükséges segédeljárás definiálása, azt külön jelezzük. Természetesen más esetben is használhat segédeljárást. Minden segédeljáráshoz írjon fejkommentet!

Nem használhat könyvtári eljárásokat, mellékhatásos beépített eljárásokat, valamint az alábbi listakezelő eljárásokat: append/3, member/2, memberchk/2, nonmember/2.

Hibakezeléssel nem kell foglalkoznia: a megírt eljárásoknak csak a fejkomment által megadott argumentumértékek esetén kell az előírt módon viselkedniük.

Az alábbi példasorban a (bináris) fa adatstruktúra alatt egy olyan Prolog kifejezést értünk, amely lehet

- levél, azaz egy `leaf' nevû struktúra, amelynek egyetlen argumentuma egy egész szám, pl. leaf(1), leaf(2); vagy
- csomópont, azaz egy node nevû kétargumentumú struktúra, amelynek mindkét argumentuma `fa', pl. node(leaf(1),leaf(2)).

A fejkommentekben használt ún. input/output módjelölések magyarázata:

```
*: az argumentum tömör bemenő, azaz nem lehet változó, és nem is
   tartalmazhat változót;
+: az argumentum bemenő, azaz nem lehet változó, de tartalmazhat változót
   (természetesen csak akkor, ha az argumentum egy struktúra);
-: az argumentum kimenő, azaz változó;
?: az argumentum lehet kimenő és bemenő is.
```

P1. Számsorozat generálása

```
% seq(+N, +M, -L): Az L lista M-N+1 hosszú, elemei 1 különbségû számtani
% sorozatot alkotnak, és L első eleme (ha van) N, ahol N és M egész számok.

| ?- seq(2, 4, L).
L = [2,3,4] ? ; no
| ?- seq(4, 2, L).
no
| ?- seq(4, 3, L).
L = [] ? ; no
| ?- seq(-4, -2, L).
L = [-4,-3,-2] ? ; no
```

P2. Számintervallum felsorolása

```
% max(+N, ?X): X egy egész szám, melyre 0 < X =< N, ahol N adott
% egész szám. Az eljárás a fenti feltételeknek megfelelő X számokat
% sorolja fel. A felsorolás sorrendjére nem teszünk megkötést.

| ?- max(1,X).
X = 1 ? ; no
| ?- max(4,X).
X = 4 ? ; X = 3 ? ; X = 2 ? ; X = 1 ? ; no
| ?- max(4,3).
yes
| ?- max(4,5).
no
| ?- max(0,X).
no
| ?- max(-6,X).
no
```

P3. Hatványozás

```
% hatv(+A, +E, -H):  H = A ^ E, ahol A egész szám, E >= 0 egész szám.

| ?- hatv(3, 5, X).
X = 243 ? ; no
```

P4. Fa csomópontjainak megszámolása

Egy fa csomópontjainak száma a benne előforduló node/2 struktúrák száma.

```
% fa_pontszama(*Fa, -N): A Fa bináris fa csomópontjainak száma N.

| ?- fa_pontszama(node(leaf(1),node(leaf(2),leaf(3))), N).
N = 2 ? ; no
| ?- fa_pontszama(node(leaf(1),node(leaf(2),node(leaf(4),leaf(3)))), N).
N = 3 ? ; no
```

P5. Fa minden levélértékének növelése

```
% fa_noveltje(*Fa0, ?Fa): Fa úgy áll elő a Fa0 bináris fából, hogy az
% utóbbi minden egyes levelében levő értéket 1-gyel megnöveljük.

| ?- fa_noveltje(node(leaf(1),node(leaf(2),leaf(3))), Fa).
Fa = node(leaf(2),node(leaf(3),leaf(4))) ? ; no
```

P6. Lista hosszának meghatározása

Egy lista hosszának az elemei számát nevezzük.

```
% lista_hossza(*Lista, -Hossz): A Lista egészlista hossza Hossz.

| ?- lista_hossza([1,3,5], H).
H = 3 ? ; no
```

P6* (szorgalmi, otthoni feladat) Lista hosszának meghatározása -- jobbrekurzív változat

```
% lista_hossza2(*Lista, -Hossz): A Lista egészlista hossza Hossz.
% Jobbrekurzív változat
```

Segédeljárás szükséges.

```
P7. Egészlista minden elemének növelése

    % lista_noveltje(*L0, ?L): Az L egészlista úgy áll elő az L0
    % egészlistából, hogy az utóbbi minden egyes elemét 1-gyel megnöveljük.

    | ?- lista_noveltje([1,5,2], L).
    L = [2,6,3] ? ; no

P8. Egy lista utolsó elemének meghatározása

    % lista_utolso_eleme(*L, ?Ertek): Az L egészlista utolsó eleme Ertek.

    | ?- lista_utolso_eleme([5,1,2,8,7], E).
    E = 7 ? ; no

P9. Egy fa leveleiben található értékek felsorolása

    % fa_levelerteke(*Fa, -Ertek): A Fa bináris fa egy levelében található
    % érték az Ertek.

    Az eljárás nemdeterminisztikus módon sorolja fel az összes
    levélértéket.  A felsorolás sorrendjére nem teszünk megkötést.

    | ?- fa_levelerteke(node(leaf(1),node(leaf(2),leaf(3))), E).
    E = 1 ? ; E = 2 ? ; E = 3 ? ; no

P10. Egy fa részfáinak a felsorolása

    Egy fa (nem feltétlenül valódi) részfájának nevezzük saját magát,
    valamint - ha a fa egy csomópont - akkor a bal és jobboldali ág részfáit.

    % fa_reszfaja(*Fa, -Resz): Resz a Fa bináris fa részfája.

    A fenti eljárás nemdeterminisztikus, azaz többféleképpen sikerül:
    a Resz változóban fel kell sorolnia a Fa összes részfáját. A felsorolás
    sorrendjére nem teszünk megkötést.

    | ?- fa_reszfaja(node(leaf(1),node(leaf(2),leaf(3))), Fa).
    Fa = node(leaf(1),node(leaf(2),leaf(3))) ? ;
    Fa = leaf(1) ? ;
    Fa = node(leaf(2),leaf(3)) ? ;
    Fa = leaf(2) ? ;
    Fa = leaf(3) ? ; no

    Gondolja meg, hogy a predikátum klózai sorrendjének változtatásakor
    hogyan változik a felsorolás sorrendje!

    A fa_reszfaja eljárás felhasználásával írja meg a 9. feladat
    megoldását, fa_levelerteke2 néven!
```

P11. Egy lista prefixumainak a felsorolása

Egy L n-elemû lista prefixumának nevezzünk egy listát, ha az az L első k elemét tartalmazza (az L-beli sorrend megtartásával), ahol 0 =< k =< n.

```
% lista_prefixuma(*L0, -L): L az L0 egészlista prefixuma.
```

A fenti eljárás nemdeterminisztikus, azaz többféleképpen sikerül: az L változóban fel kell sorolnia a L0 összes prefixumát. A felsorolás sorrendjére nem teszünk megkötést.

```
| ?- lista_prefixuma([1,4,2], Sz).
Sz = [1,4,2] ? ;
Sz = [1,4] ? ;
Sz = [1] ? ;
Sz = [] ? ; no
```

Gondolja meg, hogy a predikátum klózai sorrendjének változtatásakor hogyan változik a felsorolás sorrendje!

```
Szorgalmi, otthoni programozási feladatok
=========================================

+1. Fa mélységének meghatározása

    Egy fa mélységén az egymásba skatulyázott node struktúrák maximális
    számát értjük.

    % fa_melysege(*Fa, ?M): A Fa bináris fa mélysége M.

    | ?- fa_melysege(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), N).
    N = 2 ? ; no
    | ?- fa_melysege(node(leaf(1),node(leaf(2),node(leaf(4),leaf(3)))), N).
    N = 3 ? ; no

    Tipp: az aritmetikai kifejezésekben megengedett a max függvény
    használata, pl  | ?- X is max(2,3)+1. ---> X = 4 ? ; no


+2. Egy fa legbaloldalibb levélértékének meghatározása

    % fa_balerteke(*Fa, ?Ertek): A Fa bináris fa legbaloldalibb levelében
    % az Ertek egész érték szerepel.

    | ?- fa_balerteke(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), E).
    E = 1 ? ; no


+3. Egy fa legjobboldalibb levélértékének meghatározása

    % fa_jobberteke(*Fa, ?Ertek): A Fa bináris fa legjobboldalibb levelében
    % az Ertek egész érték szerepel.

    | ?- fa_jobberteke(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), E).
    E = 3 ? ; no


+4. Egy fa rendezettségének eldöntése

    Egy fát rendezettnek mondunk, ha a levelek értékei, balról jobbra
    haladva szigorúan monoton növő sorozatot alkotnak.

    % fa_rendezett(*Fa): A Fa bináris fa rendezett.

    | ?- fa_rendezett(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3)))).
    no
    | ?- fa_rendezett(node(node(leaf(1),leaf(3)),
                           node(leaf(5),node(leaf(6),leaf(9))))).
    yes

    A megoldásban célszerû az előző két feladat eljárásait segédeljárásként
    felhasználni, így nem szükséges további segédeljárást definiálni.

    Keressen hatékonyabb megoldást is, amelyben a fastruktúrát csak egyszer
    járja be! Ebben feltételezheti, hogy a fa csak pozitív számokat
    tartalmaz (fa_rendezett2). Ehhez a megoldáshoz szükség lehet egy
    megfelelő segédeljárásra.
```

```
+5. Fa tükörképének képzése

    % fa_tukorkepe(*Fa0, ?Fa): Fa a Fa0 bináris fa tükörképe.

    | ?- fa_tukorkepe(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), Fa).
    Fa = node(node(leaf(3),leaf(2)),node(leaf(4),leaf(1))) ? ; no
    | ?- fa_tukorkepe(node(node(leaf(1),leaf(4)),node(leaf(4),leaf(1))), Fa).
    Fa = node(node(leaf(1),leaf(4)),node(leaf(4),leaf(1))) ? ; no

+6. Fa tükörszimmetrikus voltának ellenőrzése

    % fa_tukros(*Fa): A Fa bináris fa tükörszimmetrikus.

    | ?- fa_tukros(node(node(leaf(1),leaf(4)),node(leaf(4),leaf(1)))).
    yes
    | ?- fa_tukros(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3)))).
    no

+7. Lista adott hosszú prefixuma

    Egy L n-elemû lista prefixumának nevezünk egy listát, ha az az L
    első k elemét tartalmazza (az L-beli sorrend megtartásával),
    ahol 0 =< k =< n.

    % prefix_length(+Whole, ?Prefix, +Length): A Whole lista
    % prefixuma a Prefix lista, amelynek hossza Length.
    | ?- prefix_length([a,b,c,d,e], Prefix, 3).
    Prefix = [a,b,c] ? ;
    no

+8. Lista adott helyen kezdődő szuffixuma

    Egy L  lista szuffixumának nevezünk egy listát, ha az az L utolsó
    valahány elemét tartalmazza, az L-beli sorrend megtartásával.

    % suffix_before(+Whole, ?Suffix, +Before): A Whole zárt végû lista
    % azon szuffixuma a Suffix lista, amelyet megelőző elemek száma
    % Before.
    | ?- suffix_before([a,b,c,d,e], Suffix, 3).
    Suffix = [d,e] ? ;
    no

+9. Részlista képzése

    % sublist(+Whole, ?Part, +Before, +Length): A Whole zárt végû
    % lista azon (folytonos) részlistája Part, amely előtt Before
    % számú elem áll és amelynek hossza Length.
    | ?- sublist([a,b,c,d,e], Part, 1, 3).
    Part = [b,c,d] ? ;
    no

+10. Legnagyobb közös osztó -- euklideszi algoritmus

    Írjon egy az euklideszi algoritmust megvalósító Prolog eljárást!

    % lnko(+A, +B, -LNKO): LNKO az A és B nem-negatív számok legnagyobb
    % közös osztója (de A és B nem lehet egyaránt 0).

    % Az A egész szám B-vel való osztásának M maradékát az `M is A mod B'
    % hívással állíthatja elő (A >=0, B > 0).

    | ?- lnko(24, 33, X).
    X = 3 ? ; no
```