

# Logické programování – použití: um. Int., databáze, sém.web

## PROLOG (programming in Logic)

Program popisuje "svět" Prologu (nevyjadřuje tok řízení = vývojový diagram výpočtu)

tvoří jej

databáze faktů a pravidel (tzv. klauzulí), které mají tvar:

**fakta:**     **predikát(arg1, arg2, ...argN).**

argumenty mohou být konstanty nebo proměnné

**pravidla:** **hlava :- tělo.** „:-“ čteme „jestliže“

hlavou je predikát s případnými argumenty, tělem je posloupnost faktů příp. vázaných konjunkcí „,“ nebo disjunkcí „;“

**predikát(argumenty) :- fakta.**

**cíle:**     **?- predikát(arg1, arg2, ...argN).**

Zápisem cíle spustíme výpočet, výsledkem jsou hodnoty proměnných cíle

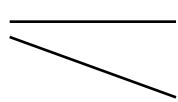
elementy programu jsou

konstanty, proměnné, struktury

společně se nazývají termy

Př.

Konstanty



čísla

atomy a, ja\_a\_ty, "NOVY", b1

Proměnné

A, \_X, \_

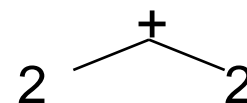
začínají velkým písmenem nebo podtržítkem. \_ je tzv. anonymní proměnná, její hodnota nás nezajímá a nevypisuje se

Struktury

muz(jan),

cte(student(jan,novy),kniha(A,Nazev))

2+2 je totéž jako +(2, 2)



Mohou být:

1) tvořené operátory a operandy

2) tvořené seznamy – výklad později

# Př. dum.pro

Jméno fce tzv funktor.  
argumenty

ma, dům, dveře jsou atomy

ma(dum, dveře).  
ma(dum, okno).  
ma(dveře, klika).  
rozbite(dveře).

fakta

?-ma(dum, dveře).  
?-ma(Neco, klika).  
?-ma(dum, Neco).  
?-ma(X, Y), rozbite(Y).

Cíle = dotazy

Odpovi yes

Odpovi Něco = dveře

Odpovi Něco = dveře

Odpovi X=dům, Y=dveře

## Postup spuštění:

V interaktivním režimu spustit Amzi Prolog, pak Listener, pak Start. Vypíše se prompt ?- Pak zavedeme program pomocí: Listener, Consult a z okna Otevřít vybereme 1dum.pro Tím Prolog zná co platí v jeho světě a můžeme se ptát na cíle. Po odpovědi odřádkujeme Při opravě programu a jeho uložení zavádíme jej pomocí Listener, Reconsult, které přepíše původní databázi faktů a pravidel. Consult by přidalo nový program za starý

# Pravidla

`cil(argumenty):-logicka formule z podcílů.`

Matematický zápis pravidla by byl  $p \leftarrow q_1 \ \& \ q_2 \ \& \ \dots \ \& \ q_n.$

## **Př. rodina.pro**

```
rodic(eva, petr). rodic(eva, jan).           %eva je rodicem petra a jana
rodic(jindrich, jan). zena(eva). muz(petr).  % vlastní jména začínají malým
muz(jan). muz(jindrich).                    % písmenem, jsou to konstanty
matka(M,D):-rodic(M,D), zena(M).
otec(O,D):- rodic(O,D), muz(O).
```

Význam pravidel:

M je matkou D, jestliže M je rodičem D a zároveň M je žena.

O je otcem D, jestliže O je rodičem D a zároveň O je mužem.

Jak definovat rodinné vztahy? Zkuste si to

```
deda(D, V) :- rodic(D, X), rodic(X, V).
```

Predikáty se odlišují dle jména i arity

**otec(O,D) je jiný predikát než otec(O)**

Prvý je vlastně binární relací mezi dvěma osobami, druhý je unární relací říkající pouze zda nějaká osoba je otcem

Anonymní proměnná = nechceme znát její hodnotu

Může jich být více v klauzuli a navzájem nesouvisí

Označuje se podtržítkem `_`

Př.

Definice otce = někdo, kdo je rodičem a mužem a nezajímá nás kdo jsou děti

**otec(O):- rodic(O,\_), muz(O).**

Def.syna zkuste sami

Databázi lze i průběžně doplňovat

# Rekurze

Definice českých panovníků – přemyslovců

Premyslovec je premysl\_orac

Premyslovec je syn premyslovce

Zapsáno prologovsky:

premyslovec(premysl\_orac).

premyslovec(P):-syn(P,R), premyslovec(R).

Alternativně to lze vyjádřit jedinou klauzulí: Přemyslovec P je buď přemysl-oráč, nebo je to syn nějakého R, který je přemyslovcem.

premyslovec(P):-

(P=premysl\_orac) ; (syn(P,R),premyslovec(R)).

Pokud nám odpověď nestačí, můžeme ji odmítnout zápisem středníku  
Prolog pak zkusí nalézt jiné řešení

1. Jak vytvořit dotaz na jména všech přemyslovců?
2. Jak definovat potomka po meči?
3. Jak definovat příbuznost osob X a Y po meči?
4. Jak definovat příbuznost osob X a Y

Ad1 Předpokládejme, že v databázi prologu máme fakta o potomcích tvaru syn(nezamysl, kresomysl) atd. Pak se lze dotazovat

?-premyslovec(P).

P= ... ;           pokud na odpověď reagujeme ; bude se hledat jiné řešení.

P= ... ;

...

no                   až se vyčerpají všechny možnosti, bude odpověď no.

Všechna řešení lze nalézt pohodlněji jediným složeným dotazem

?-premyslovec(P), write(P), fail.

který váže konjunkcí tři podcíle: nalezení přemyslovce P, výpis P, a vždy nesplněného predikátu fail, který způsobí hledání jiného řešení, které samozřejmě zase skončí fail, ale vypíše nám další jméno.

# Princip rezoluce aneb jak Prolog hledá řešení

Předpokládejme pravidla  $a$  ,  $b$  tvaru

$a :- a_1, a_2, \dots, a_n.$

$b :- b_1, b_2, \dots, b_m.$

Nechť  $b_i$  je  $a$

Pak rezolucí je

$b :- b_1, b_2, \dots, b_{i-1}, a_1, a_2, \dots, a_n, b_{i+1}, \dots, b_m.$

Tzn. když se při plnění cílů z těla pravidla  $b$  narazí na zpracování cíle  $b_i$  alias  $a$ , začnou se zpracovávat cíle těla pravidla  $a$ .

Jednotlivé cíle jsou predikáty, které Prolog porovnává s klauzulemi v jeho databázi. Proces porovnávání, když dopadne úspěšně, se nazývá **unifikací**



# Unifikace

- porovná-li se volná proměnná s konstantou, naváže se na tuto konstantu,
- porovnají-li se dvě volné (neinstalované) proměnné, stanou se synonymy,
- porovná-li se volná proměnná s termem, naváže se na tento term,
- porovnají-li se termy, které nejsou volnými proměnnými, musí být pro úspěšné porovnání stejné.

Př. unifikace v dotazu  $?- X = Y, Y = a.$  Pak  $X$  i  $Y$  má hodnotu  $a$

Pozor, operátorem „=„ unifikujeme, nepřirazuujeme, pro přiřazení slouží operátor „is“

## Př.3nsd.pro největšího společného dělitele

1. Největší společný dělitel A a A je A

2. Největší společný dělitel A a B je NSD jestliže

při A větším než B platí: A1 je A-B a největší společný dělitel A1 a B je NSD

3. při A menším než B platí B1 je B-A a největší společný dělitel B1 a A je NSD

nsd(A,A,A). % 1

nsd(A,B,NSD) : - A>B, % 2

A1 is A-B,  
nsd(A1,B,NSD).

nsd(A,B,NSD) : - A<B, % 3

B1 is B-A,  
nsd(B1,A,NSD).

Po zkontrolování souboru s programem spustíme výpočet např.

?-nsd(16,12,X).

## Př.4 fakt.pro Výpočet faktoriálu

Faktoriál 1 je 1.

Faktoriál N je F, jestliže platí, že nějaké M má hodnotu N-1 a současně faktoriál M je G a současně F má hodnotu  $G * N$

fakt(1,1).

fakt(N,F) :- M is N-1,

    fakt(M,G),

    F is G \* N.

Výpočet spustíme dotazem např.

?-fakt(3,X).

Pokud bychom výsledek odmítli vznikne chyba přeplnění stacku.

Objasněte proč?

# Zásady při plnění cílů

- Dotaz může být složen z několika cílů.
- Při konjunkci cílů jsou cíle plněny postupně zleva.
- Pro každý cíl je při jeho plnění prohledávána databáze od začátku.
- Při úspěšném porovnání klauzule s cílem je její místo v databázi označeno ukazatelem. Každý z cílů má vlastní ukazatel.
- Při úspěšném porovnání cíle s hlavou pravidla, pokračuje výpočet plněním cílů zadaných tělem pravidla.
- Cíl je splněn, je-li úspěšně porovnán s faktem databáze, nebo s hlavou pravidla databáze a jsou splněny podcíle těla pravidla.
- Není-li během exekuce některý cíl splněn ani po prohlédnutí celé databáze, je aktivován mechanismus návratu.
- Splněním jednotlivých cílů dotazu je splněn globální cíl a systém vypíše hodnoty proměnných zadaných v dotazu.
- Zjistí-li se při výpočtu, že globální cíl nelze splnit, je výsledkem no.

# Mechanismus návratu

- exekuce se vrací k předchozímu splněnému cíli, zruší se instalace (navázání) proměnných a pokouší se opětovně splnit tento předchozí cíl prohledáváním databáze dále od ukazatele pro tento cíl,
- splní-li se opětovně tento cíl, pokračuje se plněním dalšího, (předtím nesplněného) vpravo stojícího cíle,
- nesplní-li se předchozí cíl, vrací se výpočet ještě více zpět (na vlevo stojící cíl).

# Shrnutí základních principů

- **Program** specifikujeme množinou klauzulí. Klauzule mají podobu faktů, pravidel a dotazu. Prolog zná pouze to, co je definované programem.
- **Fakt** je jméno relace a argumenty (objekty) v daném uspořádání. Uspořádání je důležité.
- **Pravidlo** vyjadřuje vztahy, které platí jsou-li splněny podmínky z těla (cíle). Hlavu tvoří vždy jen jeden predikát.
- **Dotaz** může tvořit jeden nebo více cílů. Cíle mohou obsahovat proměnné i konstanty, Prolog najde tolik odpovědí kolik je požadováno (pokud existují).
- **Proměnná** je v klauzuli obecně kvantifikována. Její platnost je omezena na klauzuli.

- **Definice predikátu je posloupnost klauzulí pro jednu relaci. Predikát může určovat vztah, databázovou relaci, typ, vlastnost, funkci. Jméno predikátu musí být atomem.**
- **Plnění cíle provádí Prolog pro nový cíl prohledáváním databáze od začátku, při opakovaném pokusu prohledáváním od naposled použité klauzule.**
- **Rekurzivní definice predikátu musí obsahovat ukončovací podmínku.**
- **Typ termu je rozpoznatelný syntaxí. Atomy a čísla jsou konstanty. Atomy a proměnné jsou jednoduchými termy. Anonymní proměnná představuje neznámý objekt, který nás nezajímá. Struktury jsou složené typy dat. Pravidlo je strukturou s funktorem :-**
- **Funktor je určen jménem a aritou**

# Unifikace termů podrobněji

Dva termy jsou úspěšně porovnány (lze také říci, že si jsou podobné), pokud

- jsou totožné nebo
- proměnné v termech lze navázat na objekty tak, že po navázání proměnných jsou tyto termy totožné.

Př.      datum( D, M, 2009)                      datum(X, 12, R)

jsou unifikovatelné: D je X, M je 12, 2009 je R

datum( D, M, 2009)      datum(X, 12, 2004)

nejsou

bod(X, Y, Z)                      datum( D, M, 2003)

nejsou

datum( D, M, 2009)      datum(X, 12)

nejsou

- Prolog vybere vždy nejobecnější možnost porovnání
- Porovnání vyjadřuje operátor      =



**Při porovnání proměnné se strukturou je nutné vyloučit případ,  
kdy se tato proměnná vyskytuje ve výrazu**

**Př.?-  $X = f(X)$ . %neproveditelné porovnání**

**Způsobí navázání  $X$  na  $f(X)$**

**na  $f(X)$**

**na  $f(X)$  ...**

**Způsobí to stack overflow**

!! Aritmetické výrazy jsou termy !!

?-  $X = +( 2, 2 )$ .                      Bude odpověď

$$X = 2 + 2$$

?-  $X = 2 + 2$  .                      Bude také odpověď

$$X = 2 + 2$$

Protože + je jménem struktury a 2 , 2 jsou argumenty

Pro vyhodnocení nutno použít predikát is

?-  $X \text{ is } +( 2, 2 )$ .

$$X = 4$$

?-  $X \text{ is } 2 + 2$  .

$$X = 4$$

# Seznamy

Seznam je rekurzivní datová struktura tvaru:

$[e_1, e_2, \dots, e_n]$ , kde  $e_i$  jsou elementy seznamu.

Elementy seznamů jsou často opět seznamy

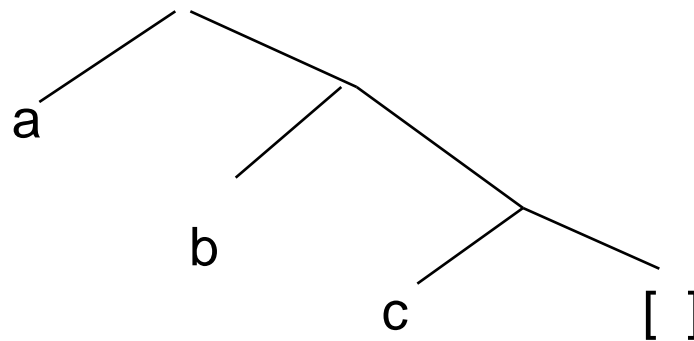
Svislínkem lze seznam rozdělit na prvý element (také se nazývá hlava) a zbytek seznamu

$[ \text{Hlava} \mid \text{Zbytek} ]$

$[ ]$  to je prázdný seznam

Př.  $[a, b, c]$  dtto  $[a \mid [b, c]]$  dtto  $[a, b \mid [c]]$  dtto  $[a, b, c \mid [ ]]$

Graficky to zachycuje obr. Všimněte si, že zbytek seznamu je opět seznam (případně prázdný seznam)



Příklady, jaké budou výsledky porovnání

[ X ] = [ a, b, c, d ] ?

no

[ X | Y ] = [ a, b, c, d ] ?

yes X=a, Y=[ b,c,d ]

[ X, Y | Z ] = [ a, b, c, d ] ?

yes X=a, Y=b, Z =[ c,d ]

[ X ] = [ [ a, b, c, d ] ] ?

yes X= [ a, b, c, d ]

[ X | Y ] = [ [ a, [ b, c ] ], d ] ?

yes X=[ a, [ b, c ] ], Y=[ d ]

[ X | Y ] = [ ] ?

no

[ X | Y ] = [ d ] ?

yes X=d, Y= [ ]

# Predikáty pro práci se seznamy

Zjištění, zda v nejvyšší úrovni seznamu existuje prvek dělá predikát  
**member(Prvek,Seznam)**

Slovně lze jeho činnost vyjádřit

member platí, 1. je-li prvek na začátku seznamu, 2. jinak member platí  
pokud prvek je ve zbytku seznamu

```
member(X,[X|_]).           %1.
```

```
member(X,[_|Y]) :- member(X,Y).      %2.
```

```
?-member(a, [b,a,c,[d,a]]).
```

yes

```
?-member(a, [b,c,[d,a]]).
```

no

Tento predikát je mezi standardními, nemusíme ho programovat

# Nalezení posledního prvku seznamu

## **last(Seznam, Prvek)**

1. Je-li v seznamu jen jeden prvek, tak je tím posledním,
2. jinak je to poslední prvek ze zbytku seznamu

last([X],X). %1.

last([\_|T],X) :- last(T,X). %2.

?- last([a,[b,c]],X).

X = [b,c]

# Odstranění prvku ze seznamu

**delete(Původníseznam, Výslednýseznam, Prvek)**

delete([X|T],T,X).

delete([Y|T],[Y|T1],X) :- delete(T,T1,X).

Zkusme formulovat slovně

Je-li prvek prvním v seznamu, je výsledkem zbytek, jinak je výsledkem seznam se stejným prvním prvkem, ale se zbytkem, v němž je vynechán uvažovaný prvek

Dotazovat se můžeme např.

?- delete([a,b,a],L,a).

L = [b,a] ;                      středníkem jsme odmítli řešení, hledá jiné

L = [a,b] ;                      znovu jsme odmítli, hledá jiné

no                                      další řešení již neexistuje

# Přidání seznamu k seznamu

## **append(Seznam1,Seznam2,Výsledek)**

append([ ],X,X).

append([A|B],X,[A|C]):-append(B,X,C).

Formulujme predikát slovně: Když přidáme k prázdnému seznamu seznam X, výsledkem bude seznam X. Když přidáme k seznamu (jehož prvý prvek je A a jeho zbytek je B) seznam X, bude výsledný seznam mít prvý prvek A a jeho zbytkem bude seznam C, který vznikne přidáním k seznamu B seznam X.

?- append([a,b],[c],X).

X = [a,b,c] a pokud teď odradkujeme (tj. odpověď nam staci), odpovi

yes

?-



Další pozoruhodnosti append

**append([ ],X,X).**

**append([A|B],X,[A|C]):-append(B,X,C).**

Zeptáme se, jaké dva seznamy musíme appendnout, aby vzniklo [a,b,c]

Prolog nám je najde a pokud budeme chtít, najde nám všechny možnosti

?- append(X,Y,[a,b,c]).

X = []

Y = [a,b,c] ;

X = [a]

Y = [b,c] ;

X = [a,b]

Y = [c] ;

X = [a,b,c]

Y = [] ;

no

?-

Námi definovaný append je obousměrný (lze zaměnit co je vstup a co výstup)

**Argumenty funkcí (tj. prologovské proměnné) mohou být bound (vázané) nebo free (volné). Zkráceně je označme b, f**

**Příklady :**

**bbb**            **?-append( [a, b], [c], [a, b, c] ).**

yes

**bbf**            **?-append( [a, b], [c], S3 ).**

S3 = [a,b,c] ;

no

**bfb**            **?-append( [a, b], S2, [a,b,c,d] ).**

S2 = [c,d] ;

no

?-

**bff**            **?-append( [a, b], S2, S3 ).**

S2 = H159

S3 = [a,b | H159] ;

no

**fbf**            **?-append( S1, [c, d], [a,b,c,d] ).**

S1 = [a,b] ;

no

**Formulujte příklady dotazů slovně!**

**fbf** ?-append( S1, [c, d], S3 ).

S1 = [ ]

S3 = [c,d] ;

S1 = [H277] H s číslem je interní proměnná Prologu

S3 = [H277,c,d] ;

S1 = [H277,H303]

S3 = [H277,H303,c,d] ;

atd

**ffb** ?-append( S1, S2, [c, d]).

S1 = [ ]

S2 = [c,d] ;

S1 = [c]

S2 = [d] ;

S1 = [c,d]

S2 = [ ] ;

no

**fff**      ?-append( S1, S2, S3 ).

S1 = []

S2 = H253

S3 = H253 ;

S1 = [H323]

S2 = H253

S3 = [H323 | H253] ;

S1 = [H323,H349]

S2 = H253

S3 = [H323,H349 | H253] ;

atd

## Vícesměrnost dalších predikátů

Predikát member je také vícesměrný

**member(X,[X|\_]).**

**member(X,[\_|Y]) :- member(X,Y).**

?- member(X,[a,[b,c],d]). **%Formulujte slovně!**

X = a ;

X = [b,c] ;

X = d ;

no

?-

## Vícesměrnost dalších predikátů

Také delete je vícesměrný predikát

**delete([X|T],T,X).**

**delete([Y|T],[Y|T1],X) :- delete(T,T1,X).**

?- delete(X,[a,b],c).

%Formulujte slovně!

X = [c,a,b] ;

X = [a,c,b] ;

X = [a,b,c] ;

no

?-

Seznamy znaků jsou řetězce. Řetězce se uzavírají do řetězcových závorek

?- [X,Y|Z]="abcd".

X = 97

Y = 98

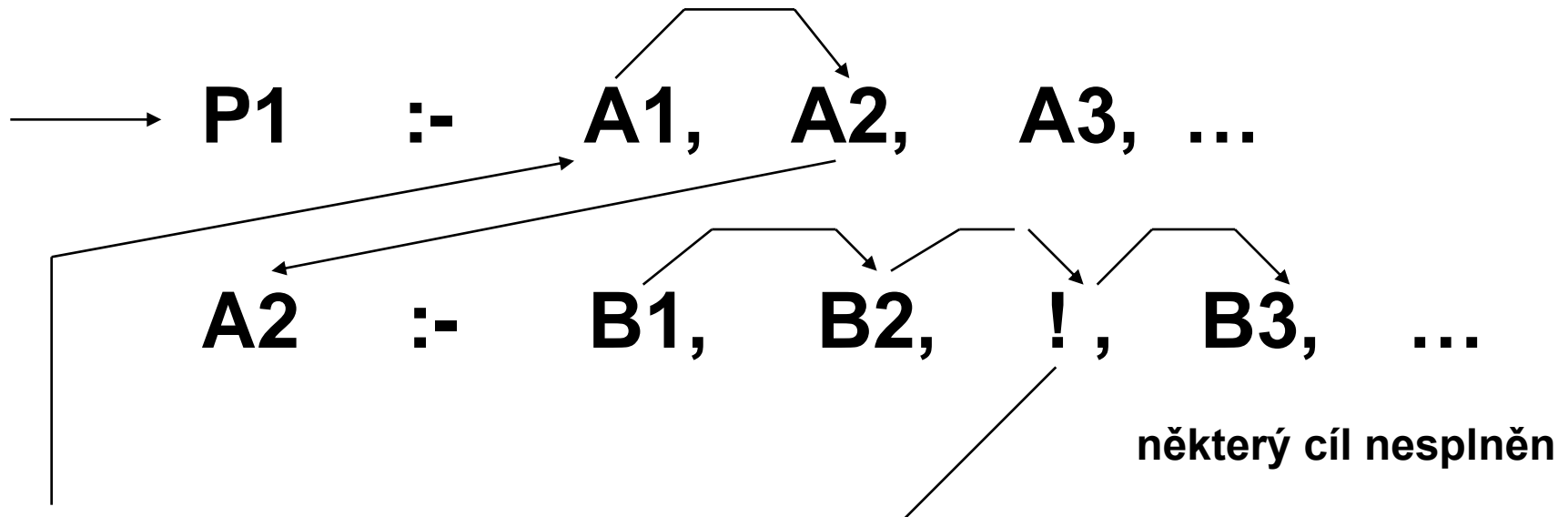
Z = [99,100]

yes

?-

# Ovlivnění mechanismu návratu

Mechanismus navracení lze ovlivnit tzv Predikátem řezu označený jako !  
Ten způsobí při nesplnění některé cíle za ním přeskok až na nové plnění cíle A1, tj způsobí nesplnění cíle A2.



**Samotný ! je vždy splněn**



# Predikát řezu

- Použijeme jej, když chceme zabránit hledání jiné alternativy
- Odřízne další provádění cílů z pravidla ve kterém je uveden
- Je bezprostředně splnitelným cílem. Projeví se pouze, když má přes něj dojít k návratu
- Změní mechanismus návratu tím, že znepřístupní ukazatele vlevo od něj ležících cílů (přesune je na konec Db)

Př. použití řezu

```
fakt(N,1) :-N=0,!.    % ! Zabrání výpočtu fakt pro záporný argument při odmítnutí výsledku
```

```
fakt(N,F) :- M is N-1,
```

```
    fakt(M,G),
```

```
    F is G * N.
```

```
?-fakt(1,X).
```

```
X=1;
```

```
no
```

# Př.Hanoiské věže

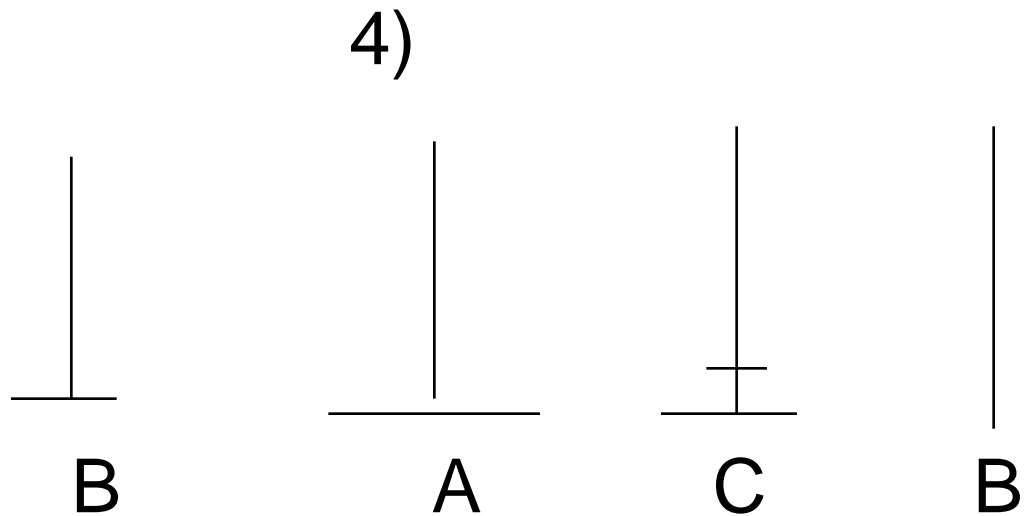
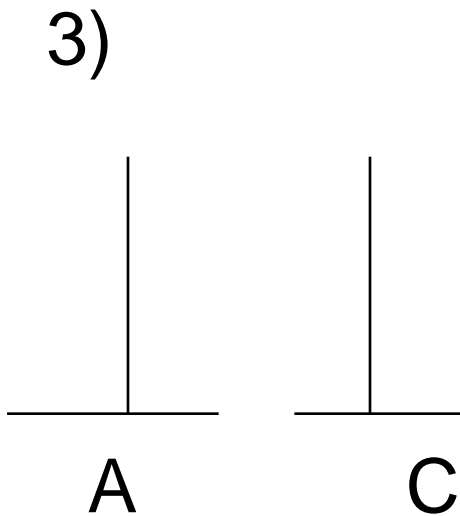
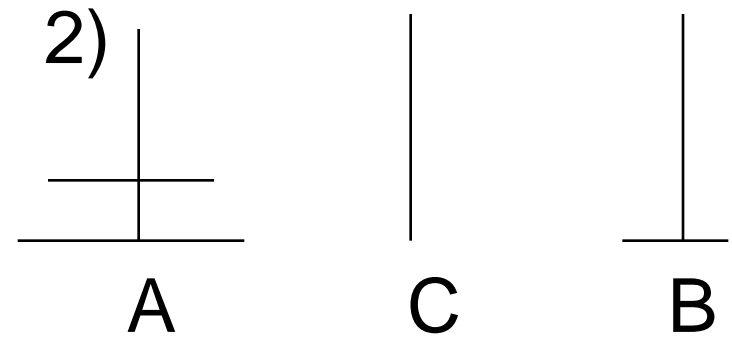
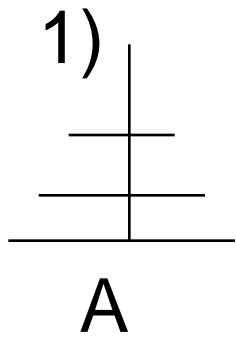
Jsou dány 3 trny A, B, C.

Na A je navléknuto N kotoučů s různými průměry tak, že vytváří tvar pagody (menší je nad větším).

Přemístěte kotouče z A na C s využitím B jako pomocného trnu tak, aby stále platilo, že nikdy není položen větší kotouč na menší

Následující obrázek ukazuje postup přemístění pro 3 kotouče

# Př.Hanoiské věže



## Př.Hanoiské věže (6Hanoi.pro)

hanoi(N) :- presun(N,levy,stred,pravy), !.

presun(0,\_,\_,\_) :- !.

presun(N,A,B,C) :- %presun z A na B za pouziti pomocného C

    M is N-1,

    presun(M,A,C,B),

    inform(A,B),

    presun(M,C,B,A).

inform(A,B) :- write([presun,disk,z,A,na,B]),

    nl.

Spustíme např.

?-hanoi(5).       nebo   ?-veze. Což způsobí výpis výzvy, přečtení počtu kotoučů, výpočet přesunů a to se opakuje až do zadání záporného počtu kotoučů. Je to i příklad, jak udělat cyklus, repeat nedělá nic, je ale libovolněkrát splnitelný, read a write nejsou při návratu znovusplnitelné (vazbu na X nelze rozvázat), fail způsobí nesplnění a tedy návrat (až k repeat)

# Standardní predikáty jazyka Prolog

Zahrnují skupiny predikátů

- I/O operace
- Řídící predikáty a testy
- Predikáty pro práci s aritmetickými výrazy
- Predikáty pro manipulaci s databází
- Predikáty pro práci se strukturami

Jazyk Prolog je vlastně tvořen resolučním mechanismem a skupinou standardních predikátů, z nichž si většinu ukážeme, ale nemusíte si všechny pamatovat.

# I/O operace

**write(X)** zápis termu do výstupního proudu

**nl** odřádkování

**tab(X)** výstup X mezer

**read(X)** čtení termu ze vstupního proudu

# Řídící predikáty a testy

<b>true</b>	vždy splněný cíl
<b>fail</b>	vždy nesplněný cíl
<b>var(X)</b>	splněno, je-li X volnou proměnnou
<b>nonvar(X)</b>	splněno, neplatí-li var(X)
<b>atom(X)</b>	splněno, je-li X instalováno na atom
<b>integer(X)</b>	splněno, je-li X instalováno na integer
<b>atomic(X)</b>	splněno, je-li X instalováno na atom nebo integer
<b>not(X)</b>	X musí být interpretovatelné jako cíl. Uspěje, pokud X není splněn
<b>call(X)</b>	X musí být interpretovatelné jako cíl. Uspěje, pokud X je splněn
<b>halt</b>	ukončí výpočet
<b>X = Y</b>	pokus o porovnání X s Y
<b>X \= Y</b>	opak =
<b>X == Y</b>	striktní rovnost
<b>X \== Y</b>	úspěšně splněn, neplatí-li ==
<b>!</b>	změna mechanismu návratu
<b>repeat</b>	nekonečněkrát splnitelný cíl
<b>X , Y</b>	konjunkce cílů
<b>X ; Y</b>	disjunkce cílů

## Predikáty pro práci s aritmetickými výrazy

<b>X is E</b>	E musí být aritm. výraz, který se vyhodnotí a porovná s X
<b>E1 + E2</b>	při instalovaných argumentech (pod. −, *, /, mod)
<b>E1 &gt; E2</b>	při instalovaných argumentech (pod. >=, <, =<, \=, =)
<b>E1 ::= E2</b>	uspěje, jsou-li si hodnoty E1, E2 rovny
<b>E1 =\= E2</b>	uspěje, nejsou-li si hodnoty E1, E2 rovny

## Predikáty k manipulaci s databází a klauzulemi

**To je již jen pro informaci**

<b>listing(X)</b>	výpis všech klauzulí na jejichž jméno je X instalováno
<b>listing</b>	výpis celého programu
<b>clause(X, Y)</b>	porovnání X a Y s hlavou a s tělem klauzule
<b>asserta(X)</b>	přidání klauzule instalované na X na začátek databáze
<b>assertz(X)</b>	totéž, ale přidává se na konec databáze
<b>retract(X)</b>	odstranění prvního výskytu klauzule X z databáze
<b>findall(X, Y, Z)</b>	všechny výskyty termu X v databázi, které splňují cíl Y jsou vloženy do seznamu Z



# Závody gymnastek –91závod.pro

Typická úloha ze sobotní přílohy novin

Urcete jména vítězek disciplin z těchto informací:

- 1) Dvorakova ani Sobotkova nevyhraly přeskok ani bradla.
- 2) V preskoku nezvitezila Vera ani Ludmila.
- 3) Sobotkova se nejmenuje ani Vera ani Jirina a kamaradi
- 4) s Beckovou
- 5) Junkova neni ani Monika ani Jirina
- 6) Vera nevyhrala na bradlech, Monika nevyhrala v prostnych.
- 7) Jednou z disciplin byla kladina.
- 8) V kazde ze ctyr disciplin zvitezila jina zavodnice.

# Závody gymnastek –91závod.pro

Čísla jednotlivých podmínek v zadání jsou v komentářích

```
vitpres(X,Y):-pojmen(X,Y),          %pozitivní cíl = důležité
    not((Y=sobotkova; Y=dvorakova)),      %1
    not((X=vera; X=ludmila)).             %2
vitbrad(X,Y):-pojmen(X,Y),
    not((Y=sobotkova; Y=dvorakova)),      %1
    not(X=vera).                          %6
vitpros(X,Y):-pojmen(X,Y),
    not(X=monika).                        %6
vitklad(X,Y):-pojmen(X,Y).               %7
```

# Závody gymnastek –91závod.pro

pojmen(X,junkova):-jmeno(X),  
X\=monika, X\=jirina.

%5

pojmen(X,sobotkova):-jmeno(X),  
X\==vera, X\==jirina.

%3

pojmen(X,beckova):- jmeno(X).

%4

pojmen(X,dvorakova):- jmeno(X).

jmeno(monika).

jmeno(jirina).

jmeno(vera).

jmeno(ludmila).

ruzne(A1,A2,A3,A4):-

A1\=A2,A1\=A3,A1\=A4,A2\=A3,A2\=A4,A3\=A4. %8

} jsou  
taková  
jména

# Závody gymnastek –91závod.pro

vitez(X1,Y1,X2,Y2,X3,Y3,X4,Y4):-

vitpres(X1,Y1), vitbrad(X2,Y2),

vitpros(X3,Y3), vitklad(X4,Y4),

ruzne(X1,X2,X3,X4), ruzne(Y1,Y2,Y3,Y4). %8

go:- vitez(JPR,PPR,JBR,PBR,JPROS, %tim se spusti  
PPROS,JKLAD,PKLAD),  
write(JPR),write(PPR),write(JBR),write(PBR),nl,  
write(JPROS),write(PPROS),  
write(JKLAD),write(PKLAD).

## Závody gymnastek (permutacemi) –92závod1.pro

vypiseme seznam jmen a seznam prijmeni vitezů  
soutezí v poradi: vitezka Bradel, vitezka Kladiny,  
vitezka Preskoku, vitezka Prostnych

```
perm([],[]).
```

```
perm(L,[X|P]) :- del(X,L,L1), perm(L1,P).
```

```
del(X,[X|T],T).
```

```
del(X,[Y|T],[Y|T1]) :- del(X,T,T1).
```

%hledá poradové číslo N, prvku E, v zadaném seznamu

```
poradi(E,[E|T],1) :-!.
```

```
poradi(E,[X|T],N) :- poradi(E,T,M), N is M+1.
```

# Závody gymnastek (permutacemi) –závod1.pro

```
%permutuj Jmena a Prijmeni (ta jsem popsal jen pocatecnimi pismeny)
vitez :- perm([j,l,m,v],J), perm([b,d,j,s],P),
    J=[J1,J2,J3,J4], P=[P1,P2,P3,P4],
% sob se nejmenuje ver    sob se nejmenuje jir
    poradi(s,P,N1),poradi(v,J,N2), N1\=N2, poradi(j,J,N3), N1\=N3,
%jun neni mon    jun neni jir
    poradi(j,P,N4),poradi(m,J,N5), N4\=N5, N4\=N3,
%bradla nevyhrala vera ani dvorak. ani sobotk
    J1\=v, P1\=d, P1\=s,
%preskok nevyhrala vera ani lud. ani dvorak, ani sobotk
    J3\=v, J3\=l, P3\=d, P3\=s,
%prostna nevyhrala monika
    J4\=m,
    write(J),write(P).
```