

Paradigmas de Linguagens de Programação
1ª Lista de Exercícios – Programação Lógica - Profa. Heloisa

- 1) Um programa Prolog tem as seguintes cláusulas:

```
% Base de Dados Geometria
vertical(seg(ponto(X,Y),ponto(X,Y1)).
horizontal(seg(ponto(X,Y),ponto(X1,Y)).
```

Dê os resultados das consultas:

```
?- vertical(seg(ponto(1,1),ponto(1,2))).
?- vertical(seg(ponto(1,1),ponto(2,Y))).
?- horizontal(seg(ponto(1,1),ponto(2,Y))).
?- vertical(seg(ponto(2,3),P)).
```

- 2) Na programação Lógica, a operação de casamento de padrão é efetuada pela unificação, que no Prolog é denotada pelo operador =. Dê o resultado das instanciações de variáveis nas operações de casamento de padrão abaixo:

- a) point(A,B) = point(1,2)
- b) point(A,B) = point(X, X, Z)
- c) [a,b,c,[ab], [], [[a], c]] = [X, Y | Z]
- d) [p, [seg], t, q] = [X,[Y],Z | S]
- e) triangle (point(-1,0),P2, P3) = triangle (P1, point(1,0), point (0,Y))
- f) [X,Y | Z] = [2,[3,4]]
- g) [lista, de, exercicios, de , IA] = [X, de | W]

- 3) Dê o resultado das seguintes consultas:

```
?- gosta_de(maria,X) = gosta_de(X,joao).
false
?- 3+5 >= 5+3.
true
?- 3+(5*2-1)/4 := 5-2.
true % As expressões são calculadas e os resultados são comparados
?- 3+(5*2-1)/4 = 5-2.
false % Esse é o operador de unificação. Os termos não unificam
?- 3+(5*2-1)/4 == 5-2.
false % Esse operador verifica se os termos são idênticos. Nesse caso, não são.
?- X=4, X is (X-2)*2.
true % Na segunda parte da consluta o valor de X não é alterado pelo is ns tem valor
% igual ao do resultado da expressão aritmética, portanto, unifica.
?- X is X+1.
Erro % Variável não instanciada não pode ser usada em expressões airtiméticas
?- X =1, X is X + 1.
false. % X instancia com 1. Na segunda parte da consulta, o valor de X não é alterado pelo is
```

- 4) Constua uma base de dados sobre livros com:

- a) pelo menos cinco estruturas do tipo exemplificado abaixo, onde a lista de palavras chave pode ter entre três e seis elementos.

```
livro(nome('C completo e total'), autor('Schildt'), pal_chave([linguagemc, programacao,
computacao])).
```

```
livro(nome('Paradigmas de LP'), autor('Schildt'), pal_chave([linguagemc, programacao,
computacao])).
```

```
livro(nome('LISP'), autor('Winston, P. ; Horn, B.'), pal_chave([lisp, programacao, list])).
```

```
livro(nome('Introdução à Programação Orientada a Objetos usando JAVA'), autor('Santos,
R.'), pal_chave([POO, classes, objetos])).
```

b) Escreva consultas para encontrar:

- nome do autor, dado o nome do livro
- nome do livro, dado o nome do autor
- as palavras chave, dado o nome do livro
- nome do autor e o nome do livro, dada uma palavra chave

```
> livro(nome('Paradigmas de LP'), autor(X),_).  
> livro(nome(X), autor('Schildt'), _).  
> livro(nome("Paradigmas de LP"), _,pal_chave(L)).  
> livro(nome(Nome), autor(Autor), pal_chave(L)), pertence (lisp, L).
```

c) Escreva um programa para, dada uma lista de palavras chave, encontrar os livros (nome e autor) que tem pelo menos uma das palavras chave fornecidas. Os livros encontrados devem ser retornados um de cada vez.

```
busca_livro(Lista, Nome, Autor):-  
    livro(nome(Nome), autor(Autor), pal_chave(Pal_chave)),  
    tem_palavra(Lista,Pal_chave).
```

```
tem_palavra([X | Y], Lista):-  
    pertence(X.Lista), !.
```

```
tem_palavra([X | Y], Lista):-  
    tem_palavra(Y, Lista).
```

5) Defina os predicados `n_par(Lista)` e `n_impar(Lista)` para determinar se uma dada lista tem número par ou ímpar de elementos, respectivamente.

```
comprimento([ ], 0) :- !.  
comprimento([X|L],N) :-  
    comprimento(L,N1), N is N1 + 1.  
n_par(L) :-  
    comprimento(L,N), N mod 2 == 0.  
N_impar(L):-  
    comprimento(L,N), N mod 2 /= 0.
```

6) Defina a relação `shift_esq(Lista1, Lista2)` tal que `Lista2` seja a `Lista1` com uma rotação à esquerda. Por exemplo,

```
?- shift_esq([1,2,3,4,5], L1).  
L1 = [2,3,4,5,1]
```

```
shift_esq([ ],[ ]):- !.  
shift_esq([X], [X]):- !.  
shift_esq([X|Y], L) :- add_ultimo(X,Y,L).  
add_ultimo(X,[ ], [X]) :- !.  
add_ultimo(X,[E|Y],[E|Z]) :- add_ultimo(X,Y,Z).
```

7) Defina a relação `shift_dir(Lista1, Lista2)` tal que `Lista2` seja a `Lista1` com uma rotação à direita. Por exemplo,

```
?- shift_dir([1,2,3,4,5], L1).  
L1 = [5,1,2,3,4]
```

8) Defina a relação `shift_n_esq(N, Lista1, Lista2)` tal que `Lista2` seja a `Lista1` com `N` rotações à esquerda. Por exemplo,

```
?- shift_n_esq(3, [1,2,3,4,5], L1).
```

L1 = [4,5,1,2,3]

- 9) Defina a relação `traduz(L1, L2)` para traduzir uma lista de números entre 0 e 9 para a palavra correspondente. Por exemplo:

?- `traduz([1, 2, 4], L).`

L = [um, dois, quatro]

Use a relação auxiliar: `t(0, zero), t(1, um), t(2, dois),t(9, nove)`

`traduz([], []):- !.`

`traduz([X|Y], [Z|W]) :- t(X,Z), traduz(Y,W).`

- 10) Defina a relação `subset(Set, Subset)` onde Set e Subset são duas listas representando conjuntos, tal que seja possível verificar se Subset é subconjunto de Set. Por exemplo:

?- `subset([1,b,c], [1,c]).`

true

`subset(Set,[]):- !.`

`subset(Set, [X|Y]) :- pertence(X, Set), subset(Set,Y).`

- 11) Defina a relação `max(X, Y, Max)` tal que Max é o maior entre os números X e Y.

`max(X,Y,X):- X >=Y, !.`

`max(X,Y,Y).`

- 12) Defina o predicado `maxlista(Lista, Max)` tal que Max seja o maior número na lista de números Lista.

Versão 1:

`maxlista([X],X) :- !.`

`maxlista([X|Y],X) :- maxlista(Y,M), X >=M, !.`

`maxlista([X|Y], M) :- maxlista(Y,M).`

Versão 2:

`maxlista([X],X) :- !.`

`maxlista([X|Y],M) :- maxlista(Y,N), max(X,N,M).`

- 13) Defina o predicado `between(N1, N2, X)` tal que, para dois números inteiros dados N1 e N2, gere uma lista com todos os inteiros que satisfaçam a restrição $N1 \leq X \leq N2$.

`between(N1,N2,L):- N1 <=N2, between1(N1, N2, L).`

`between1(N,N,[N]):- !.`

`between1(N1, N2, [N1|X]) :- Nnovo is N1 + 1, between1(Nnovo,N2,X).`

- 14) A relação a seguir classifica números em tres classes: positivo, negativo e zero:

`classe (N, positivo) :- N > 0.`

`classe (0, zero).`

`classe (N, negativo) :- N < 0.`

Defina esse procedimento de maneira mais eficiente usando o corte.

`classe (N, positivo) :- N > 0, !.`

`classe (0, zero):- !.`

`classe (N, negativo).`

- 15) Defina o procedimento `split(Numeros, Positivos, Negativos)` que divide uma lista de números em duas listas: uma contendo os números positivos (incluindo o zero) e outra contendo os negativos. Proponha duas versões: uma usando corte e outra sem uso do corte.

```
split([ ], [ ], [ ]):- !.  
split([X|Cauda], [X|Y], Z) :- X>=0, split(Cauda,Y,Z), !.  
split([X|Cauda], Y, [X|Z]) :- split(Cauda, Y, Z).
```

- 16) Listas podem ser usadas para representar conjuntos. Implemente procedimentos em prolog para realizar as operações básicas entre conjuntos: união, interseção e diferença. Note que quando listas são usadas para representar conjuntos, elementos duplicados não podem aparecer nas listas.

```
uniao([ ], Y, Y):- !.  
uniao([X|Cauda], Y, [X|Z]) :- not(member(X,Y)), uniao(Cauda,Y,Z),!.  
uniao([X|Cauda], Y, Z) :- uniao(Cauda,Y,Z).
```

```
inter([ ], _, [ ]):- !.  
inter([X|Y], Z, [X|W]) :- member(X,Z), inter(Y,Z,W), !.  
inter([X|Y], Z, W):- inter(Y,Z,W).
```

```
difer([ ], L, [ ]):- !.  
difer([X|Y], L, [X|Z]) :- not (member(X,L)), difer(Y,L,Z),!.  
difer([X|Y], L, Z) :- difer(Y,L,Z).
```

- 17) Construir um programa Prolog para o mesmo problema do exercício anterior, sem uso do corte.
- 18) Fazer um programa Prolog para, dadas 3 listas representando conjuntos de números, construir a interseção dessas listas: lista contendo somente os elementos que aparecem nas três listas dadas. Assumir que as listas dadas tem apenas números, sem repetições. Os elementos da interseção podem aparecer em qualquer ordem na lista resultante. Usar o corte (!) para omitir condições.

```
inter_tres(L1, L2, L3, L) :- inter(L1,L2,Linter), inter(Linter,L3, L).  
inter([ ], _, [ ]):- !.  
inter([X|Y], Z, [X|W]) :- member(X,Z), inter(Y,Z,W), !.  
inter([X|Y], Z, W):- inter(Y,Z,W).
```

- 19) Construir um programa Prolog para verificar se um dado elemento pertence a uma lista, em qualquer nível. (Use o predicado `is_list(X)`, que retorna true se o seu argumento for lista.)

```
encontra_atomo(X, [X]) :- !.  
encontra_atomo(X, [X|Y]) :- !.  
encontra_atomo(X, [Y|Z]) :- is_list(Y), encontra_atomo(X,Y), !.  
encontra_atomo(X, [Y|Z]) :- encontra_atomo(X,Z).
```

- 20) Construir um programa Prolog para, dada uma lista numérica, fazer a ordenação dos elementos dessa lista.

```
ordena([ ], [ ]) :- !.  
ordena(L, [Min|R]) :- minlista(L,Min), retira(Min,L,L1), ordena(L1, R).  
  
minlista([X],X):- !.  
minlista([X|Y], X) :- minlista(Y,M), X <= M, !.  
minlista([X|Y], M) :- minlista(Y,M).  
  
retira(X, [ ], [ ]) :- !.  
retira(X, [X|Y], Y) :- !.  
retira(X, [Y|Z], [Y|W]) :- retira(X,Z,W).
```

- 21) Escreva um programa Prolog para, dada uma lista numérica em ordem crescente e um número, inserir o número dado na posição correta para manter a ordenação.

```
insere_em_ordem(E, [], [E]) :- !.  
insere_em_ordem(E, [X|Y], [E,X|Y]) :- E <= X, !.  
insere_em_ordem(E, [X|Y], [X|Z]) :- insere_em_ordem(E, Y, Z), !.
```

- 22) Escreva um programa Prolog para, dada uma lista L e dois elementos, E1 e E2, substituir todas as ocorrências de E1 por E2 em L, no primeiro nível da lista.
- 23) Escreva um programa Prolog para, dada uma lista L e dois elementos, E1 e E2, substituir todas as ocorrências de E1 por E2 em L, em todos os níveis da lista.
- 24) Escreva um predicado *troca_pos(E1, E2, Lin, Lout)* em Prolog para, dados dois elementos *E1* e *E2* e uma lista *Lin* com qualquer número de elementos, trocar os elementos *E1* e *E2*, ou seja, substituir as ocorrências de *E1* por *E2* e substituir as ocorrências de *E2* por *E1*, no primeiro nível da lista dada.

```
troca_pos(E1, E2, [], []) :- !.  
troca_pos(E1, E2, [E1], [E2]) :- !. %não é obrigatória  
troca_pos(E1, E2, [E2], [E1]) :- !. %não é obrigatória  
troca_pos(E1, E2, [E1|Y], [E2|Z]) :- troca_pos(E1, E2, Y, Z), !.  
troca_pos(E1, E2, [E2|Y], [E1|Z]) :- troca_pos(E1, E2, Y, Z), !.  
troca_pos(E1, E2, [E|Y], [E|Z]) :- troca_pos(E1, E2, Y, Z).
```

- 25) Construir um procedimento em PROLOG para, dado um elemento e uma lista, possivelmente com sublistas, contar quantas vezes esse elemento aparece na lista, em todos os níveis.

```
conta_todos(X, [], 0) :- !.  
conta_todos(X, [X|Y], N) :- conta_todos(X, Y, N1), N is N1+1, !.  
conta_todos(X, [Y|Z], N) :- is_list(Y), conta_todos(X, Y, N1),  
                             conta_todos(X, Z, N2), N is N1+N2, !.  
conta_todos(X, [Y|Z], N) :- conta_todos(X, Z, N).
```

- 26) Construir um procedimento PROLOG para, dada uma lista, possivelmente com listas aninhadas, desparentizar a lista dada. (Desparentizar significa construir uma lista com os mesmos átomos que a lista original, sem listas internas. Use o predicado *is_list(X)*, que retorna true se o seu argumento for lista.)
- 27) Construir um procedimento em PROLOG para, dada uma lista que contém pares de elementos (lista de dois elementos) construir uma lista com os mesmos pares elementos invertendo a ordem dos dois elementos do par. Por exemplo, dada a lista *[[a, b], [2000, nome('Joao')], [x, [v, f, 3, a]]]* o resultado deve ser *[[b, a], [nome('Joao'), 2000], [[v, f, 3, a], x]]*.

```
inverte_pares([], []) :- !.  
inverte_pares([X|Y], [Z|W]) :- troca_par(X, Z), inverte_pares(Y, W).  
  
troca_par([A, B], [B, A]).
```

- 28) Defina um programa em Prolog para resolver o problema abaixo, usando a estratégia “gerar e testar” (Fonte: <https://programmingpraxis.com/2009/06/16/who-owns-the-zebra/>)

Este famoso puzzle foi publicado na revista Life a 17 de Dezembro, 1962. Foi atribuído várias vezes a Albert Einstein e Lewis Carroll, mas o verdadeiro autor não é conhecido.

Há várias versões, esta é a original da revista:
1. Existem 5 casas.

2. O Inglês mora na casa vermelha
3. O Espanhol tem um cão.
4. O café é bebido na casa verde.
5. O Ucraniano bebe chá.
6. A casa verde está imediatamente à direita da casa de marfim.
7. O fumador de Old Gold é dono de caracóis.
8. Kools são fumados na casa amarela.
9. Leite é bebido na casa do meio.
10. O Norueguês vive na primeira casa.
11. O homem que fuma Chesterfields vive na casa ao lado do homem com a raposa.
12. Kools são fumados na casa ao lado daquela onde o cavalo é guardado.
13. O fumador de Lucky Strike bebe sumo de laranja.
14. O Japonês fuma Parliaments.
15. O Norueguês vive ao lado da casa azul.

Assuma que cada uma das 5 casas está pintada com uma cor diferente, os seus habitantes são de diferentes nacionalidades, donos de diferentes animais, bebem bebidas diferentes e fumam marcas diferentes de cigarros.

Encontre a resposta para as perguntas: Quem bebe água? Quem é dono da zebra?

- 29) Defina um programa em Prolog para resolver o problema abaixo, usando a estratégia “gerar e testar” (Fonte: <http://www.ebah.com.br/content/ABAAAhE9cAA/prolog-1001-exercicios-resolvidos>):
- Três músicos de uma banda multinacional executam um solo em um trecho de música. Cada um toca uma única vez. O pianista toca primeiro. John toca saxofone e toca antes do australiano. Mark é dos Estados Unidos e toca antes do violinista. Um solista vem do Japão e um se chama Sam. Encontre quem vem de qual país, quem toca qual instrumento e em qual ordem.