FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

EXERCÍCIOS DE PROGRAMAÇÃO EM LÓGICA

LUÍS PAULO REIS
DANIEL CASTRO SILVA

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO

PROGRAMAÇÃO EM LÓGICA - 3º ANO

SETEMBRO DE 2007

Exercício LIST 1. Funcionamento das Listas [H|T]

Preveja os resultados das seguintes questões em Prolog:

```
a) ?- [a|[b,c,d]] = [a,b,c,d].
b) ?- [a|b,c,d] = [a,b,c,d].
c) ?- [H|T] = [apple, broccoli, refrigerator].
d) ?- [H|T] = [a, b, c, d, e].
e) ?- [H|T] = [apples, bananas].
f) ?- [H|T] = [a, [b,c,d]].
g) ?- [H|T] = [apples].
h) ?- [H|T] = [].
i) ?- [One, Two | T] = [apple, sprouts, fridge, milk].
j) ?- [X,Y|T] = [a|Z].
k) ?- [H|T] = [apple, Z].
l) ?- [a|[b|[c|[d|[]]]]] = [a,b,c,d].
```

Solução:

a) yes

b) no

```
c) H = apple
T = [broccoli, refrigerator]
d) H = a
T = [b, c, d, e]
e) H = apples
T = [bananas]
f) H = a
T = [[b, c, d]]
g) H = apples
T = []
h) no
i) One = apple
Two = sprouts
T = [fridge, milk]
```

j) `X = a`

`Y = _01`

`T = _03`

`Z = [_01 | _03]`

k) `H = apple`

`T = [_01]`

`Z = _01`

l) yes

Exercício LIST 2. Funcionamento das Listas [H|T]

Resolva as igualdades, dizendo quais os valores finais das variáveis.

```
a) ?- lista([a,[b],c,[d]]) = lista([_|[X|X]]).
b) ?- lista([[a],[b],C])=lista([C,B,[a]]).
c) ?- lista([c,c,c])=lista([X|[X|_]]).
d) ?- lista([a,[b,c]])=lista([A,B,C]).
e) ?- [joao,gosta,peixe]=[X,Y,Z].
f) ?- [gato]= lista([X|Y]).
g) ?- [vale,dos,sinos])=[sinos,X,Y].
h) ?- [branco,Q]=[P,cavalo].
i) ?- [1,2,3,4,5,6,7]=[X,Y,Z|D].
```

Exercício LIST 3. Concatenação de Listas - Predicado Append

Implemente o predicado append(L1,L2,L) em que L é constituída pela concatenação das listas L1 e L2.

Solução:

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

Exercício LIST 4. Inversão de Listas

Construa um predicado *inverter(L1, L2)* que calcule a lista invertida de uma dada lista.

Solução:

```
inverter(Lista,InvLista):-
      rev(Lista,[],InvLista).
rev([H|T],S,R):-
      rev(T,[H|S],R).
rev([],R,R).
```

Exercício LIST 5. Membros de uma Lista - Predicado Member e Last

a) Implemente o predicado membro(X,L) que sucede se X for um membro da lista L. (member)

b) Utilizando unicamente o predicado append uma só vez, implemente o predicado membro(X,L). (member)

c) Utilizando unicamente o predicado append uma só vez, implemente o predicado last(L,X) que retorna a último elemento de uma lista.

d) Implemente um predicado que determine o n-ésimo membro de uma lista.

Solução:

```
b) membro(X,L):- append(_,[X|_],L).
c) last(L,X):- append(_,[X],L).
d)
nth_membro(1,[M|_],M).
nth_membro(N,[_|T],M):-
      N>1,
      N1 is N-1,
      nth_membro(N1,T,M).
```

Exercício LIST 6. Remover Elementos de Listas - Predicados Delete

a) Utilizando unicamente o predicado append duas vezes, implemente o predicado *delete_one(X,L1,L2)* que remove uma ocorrência de um item numa lista.

b) Implement um predicado *delete_all(X,L1,L2)* que remova todas as ocorrências de um elemento numa lista

c) Implemente um predicado que remove as ocorrências dos elementos que se encontram numa lista *LX*, numa outra lista *L1*, *delete_all_list(LX, L1, L2).*

Solução:

```
a) delete_one(X,L,L1):-
      append(La,[X|Lb],L),
      append(La,Lb,L1).
```

Exercício LIST 7. Elementos de Listas - Predicado Before

a) Utilizando unicamente o predicado append duas vezes, implemente o predicado before. O predicado before tem três argumentos e sucede se os dois primeiros argumentos forem membros da lista que constitui o terceiro argumento, e o primeiro argumento ocorrer na lista antes do segundo.

Solução:

```
before(A,B,L):-
      append(_,[A|L1],L),
      append(_,[B|_],L1).
```

Exercício LIST 8. Contagem de Elementos de Listas

a) Defina o predicado *conta(Lista, N)* que sucede se a Lista tem N elementos.

b) Defina o predicado *conta_elem(X, Lista, N)* que sucede se a Lista tem N elementos com o valor X.

Exercício LIST 9. Substituição e Eliminação de Elementos de Listas

a) Defina o predicado *substitui(X,Y,Lista1,Lista2)* que subsitui todas as ocorrências de X em Lista1 por Y, resultando Lista2.

b) Defina o predicado *elimina_duplicados(Lista1,Lista2)* que elimina os duplicados em Lista1, resultando Lista2.

Exercício LIST 10. Ordenação de Listas

a) Defina o predicado ordenada(Lista) que é verdadeiro se Lista é uma lista de inteiros ordenada.

b) Defina o predicado ordena(L1, L2) que ordena a lista L1, tendo como resultado L2.

Solução:

```
a)
ordenada([N]).
ordenada([N1,N2]):- N1 =< N2.
ordenada([N1,N2|Resto]):-
      N1 =< N2,
      ordenada([N2|Resto]).
```

Exercício LIST 11. Achatar Listas

Defina a relação achata_lista(Lista,ElemsLista) em que Lista é uma lista eventualmente de listas e ElemsLista é uma lista com todos os elementos de Lista ao mesmo nível.

Exemplo:

```
?- achata_lista([a, [1,2,3], b],L).
L=[a,1,2,3,b]
```

Solução:

```
achata_lista([],[]).
achata_lista(X,[X]):- atomic(X).
achata_lista([Cab|Rest],L):-
      achata_lista(Cab,L1),
      achata_lista(Rest,L2),
      append(L1,L2,L).
```

Exercício LIST 12. Calcular Permutações

Construa um predicado *permutacao(L1,L2)* que resulte se *L2* for uma permutação de *L1*.

Exercício LIST 13. Listas de Números.

a) Construa um predicado *lista_ate(N,L)* que devolva a lista L de todos os números inteiros entre 1 e N.

b) Construa um predicado *lista_entre(N1,N2,L)* que devolva a lista L de todos os números inteiros entre N1 e N2 (ambos incluídos).

c) Construa um predicado *soma_lista(L, Soma)*, que some todos os elementos da lista L, obtendo como resultado Soma.

d) Escreva um predicado *par(N)* que dado um número inteiro N, determine se ele é ou não um número par.

e) Escreva um predicado *lista_pares(N, Lista)* que aceite um número inteiro e que determine a lista de todos os números pares iguais ou inferiores a esse número.

f) Escreva um predicado *lista_impares(N, Lista)* que aceite um número inteiro e que determine a lista de todos os números impares iguais ou inferiores a esse número.

Exercício LIST 14. Números Primos

a) Escreva um predicado *primo(N)* que dado um número inteiro N, determine se ele é ou não um número primo (um número primo é aquele que só é divisível por 1 e por ele próprio).

b) Escreva um predicado *lista_primos(N, Lista)* que aceite um número inteiro e que determine a lista de todos os números primos iguais ou inferiores a esse número.

Exercício LIST 15. Produto Interno de Vectores

Construa um predicado *produto_interno (L1, L2, N)* que calcule o produto interno das listas L1 e L2.

Exercício LIST 16. Predicado Mistério

Considere a seguinte definição do predicado mistério:

```
misterio([],[]).
misterio([X],[X]).
misterio([X,Y|L],[X,censurado|M]):- misterio(L,M).
```

a) Descreva o que o predicado faz sobre uma lista constituída por palavras.

b) É comum as definições recursivas de manipulação de listas terem unicamente uma condição de base. Porque necessita a definição acima de duas condições de base ?

Exercício LIST 17. Listas Palindromas

a) Recorrendo ao predicado anterior, implemente um predicado que determine se uma lista é um palindroma. Nota: um palindroma pode ser lido da mesma forma para a frente ou para trás (exemplo: [x,a,m,a,x]).

b) Implemente o mesmo predicado sem recorrer ao predicado inverter.

Solução:

```
a) palindroma(L):- inverter(L,L).
```

Exercício LIST 18. Duplicação de Elementos de uma Lista

a) Construa um predicado que duplique os elementos de uma lista

b) Construa um predicado que copie os elementos de uma lista N vezes para a lista resultado

Exemplo a):

```
?- duplicar([a,b,c,c,d],X).
X = [a,a,b,b,c,c,c,c,d,d]
```

Exemplo b):

```
?- duplicarN([a,b,c],3,X).
X = [a,a,a,b,b,b,c,c,c]
```

Exercício LIST 19. Run-Legth Encoding

a) Crie um predicado que faça a compressão run-length encoding de uma lista.

b) Modifique o programa de forma a que se um elemento não tiver duplicados, é simplesmente copiado para a lista resultado.

c) Construa as versões de descompressão de listas, codificadas em run-lenght nas duas alíneas anteriores.

Exemplo a):

```
?- runlength([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).
X = [[4,a],[1,b],[2,c],[2,a],[1,d][4,e]]
```

Exemplo b):

```
?- run_lenght_modificado([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).
X = [[4,a],b,[2,c],[2,a],d,[4,e]]
```

Exercício LIST 20. Eliminar Elementos de N em N numa lista

Crie um predicado dropN que elimine elementos de N em N numa lista

Exemplo:

```
?- dropN([a,b,c,d,e,f,g,h,i,k],3,X).
X = [a,b,d,e,g,h,k]
```

Solução:

```
dropN(L1,N,L2) :- drop(L1,N,L2,N).

drop([],_,[],_).
drop([_|Xs],N,Ys,1) :- drop(Xs,N,Ys,N).
drop([X|Xs],N,[X|Ys],K):- K > 1, K1 is K - 1, drop(Xs,N,Ys,K1).
```

Exercício LIST 21. Extrair uma fatia de uma Lista

Crie um predicado slice(Lista, Ind1, Ind2, Result) que extraia uma fatia de uma lista desde o índice Ind1 até ao índice Ind2 (ambos os extremos incluídos).

Exemplo:

```
?- slice([a,b,c,d,e,f,g,h,i,k],3,7,L).
X = [c,d,e,f,g]
```

Exercício LIST 22. Rotação de uma Lista

Construa um predicado que rode de N elementos para a esquerda uma lista.

Exemplos:

```
?- rodar([a,b,c,d,e,f,g,h],3,X).
X = [d,e,f,g,h,a,b,c]
?- rodar([a,b,c,d,e,f,g,h],-2,X).
X = [g,h,a,b,c,d,e,f]
```

Exercício LIST 23. Sorteios Aleatórios e Listas

a) Extraia um determinado número de elementos seleccionados aleatoriamente de uma lista. E construa uma nova lista com esses elementos

b) Sorteie N elementos entre 1 e M aleatoriamente e coloque-os numa lista.

c) Gere uma permutação aleatória dos elementos de uma lista

Exemplo a):

?- rnd_selectN([a,b,c,d,e,f,g,h],3,L).

L = [e,d,a]

Exemplo b):

?- rnd_select(6,49,L).

L = [23,1,17,33,21,37]

Exemplo c):

?- rnd_permutation([a,b,c,d,e,f],L).

L = [b,a,d,c,e,f]

Exercício LIST 24. Bubble Sort de Listas

Implemente o conhecido método de ordenação bubble sort para listas em Prolog.

Solução:

```
bubble_sort(List,Sorted):-
        b_sort(List,[],Sorted).

b_sort([],Acc,Acc).
b_sort([H|T],Acc,Sorted):-
        bubble(H,T,NT,Max),
        b_sort(NT,[Max|Acc],Sorted).

bubble(X,[],[],X).
bubble(X,[Y|T],[Y|NT],Max):-
        X > Y,
        bubble(X,T,NT,Max).
bubble(X,[Y|T],[X|NT],Max):-
        X <= Y,
        bubble(Y,T,NT,Max).
```

Exercício LIST 25. Triângulo de Pascal

Defina o predicado pascal(N, L), onde L é a Nésima linha do triângulo de Pascal:

```
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
```