Prolog para todas as ocasiões: 1001 exercícios resolvidos de programação em lógica usando SWI-Prolog

Alexsandro Santos Soares
Marcelo Cupertino

© *Versão preliminar desenvolvida em 9 de dezembro de 2010*

Sumário

Sumário **1**

1 Processamento simbólico **3**
1.1 Árvore genealógica 3
1.2 Desafios lógicos 8
1.3 Recursão 18

2 Exercícios Diversos **25**
2.1 Teoria dos Números 25
2.2 Listas 28
2.3 Algoritmos baseados em relações de recorrência 59
2.4 Lógica 61
2.4.1 Formas Normais 62
2.4.2 Unificação 65
2.5 Estatística e Probabilidade 70
2.6 Tempo e Datas 74

Referência Bibliográfica **79**

1 Processamento simbólico

1.1 Árvore genealógica

Ex. 1 — (Sexo dos personagens da série *Os Simpsons*) Escrever uma banco de dados sobre o sexo de cada personagem integrante da família Simpson e também sobre a relação de progenitor.

Solução:

```
% Fatos.

homem( abraham).
homem( clancy).
homem( herbert).
homem( homer).
homem( bart).

mulher( mona).
mulher( jacqueline).
mulher( abbie).
mulher( marge).
mulher( patty).
mulher( selma).
mulher( lisa).
mulher( maggie).
mulher( ling).

progenitor( abraham, homer).
progenitor( abraham, abbie).
progenitor( abraham, herbert).
progenitor( mona, homer).

progenitor( clancy, marge).
progenitor( clancy, patty).
progenitor( clancy, selma).
progenitor( jacqueline, marge).
progenitor( jacqueline, patty).
progenitor( jacqueline, selma).
```

```
progenitor( homer, bart).
progenitor( homer, lisa).
progenitor( homer, maggie).
progenitor( marge, bart).
progenitor( marge, lisa).
progenitor( marge, maggie).

progenitor( selma, ling).
```

Uso:

SWI

```
?- mulher(Quem). % Quem são as mulheres da família?
Quem = mona ;
Quem = jacqueline ;
Quem = abbie ;
Quem = marge ;
Quem = patty ;
Quem = selma ;
Quem = lisa ;
Quem = maggie ;
Quem = ling.

?- progenitor(Quem,selma). % Quem são os progenitores de Selma?
Quem = clancy ;
Quem = jacqueline ;
false.
```

Comentário

O banco de dados desenvolvido aqui será reutilizado nos exercícios seguintes.

Ex. 2 — (Relações de parentesco: pai, mãe, filho, filha e irmãos) Como determinar quem é pai, mãe, filho, filha ou irmãos em uma família?

Solução:

```
pai( P, F):- homem( P), progenitor( P, F).
mae( M, F):- mulher( M), progenitor( M, F).

filho( F, P):- homem( F), progenitor( P, F).
filha( F, P):- mulher( F), progenitor( P, F).
```

```
irmaos( A, B):-
   progenitor( P, A),
   progenitor( P, B),
   A\== B.
```

Uso:

SWI

```
?- pai(Pai,Quem). % Quem é o pai de quem?
Pai = abraham,
Quem = homer ;
Pai = abraham,
Quem = abbie ;
Pai = abraham,
Quem = herbert ;
Pai = clancy,
Quem = marge .

?- filho(bart,Quem). % De quem Bart é filho?
Quem = homer ;
Quem = marge .

?- irmaos(A,B). % Quais são os irmãos?
A = homer,
B = abbie ;
A = homer,
B = herbert ;
A = abbie,
B = homer ;
A = abbie,
B = herbert ;
A = herbert,
B = homer .
```

Comentário

Na relação `irmaos` foi necessário a utilização do predicado `\==`, que pode ser lido como *não idêntico a*, pois ninguém pode ser irmão de si mesmo.

Ex. 3 — (Relações de parentesco: tio, tia, primo, prima, avô e avó) Como determinar quem é tio, tia, primo, prima, avô ou avó em uma família?

Solução:

```
tio( T, S) :-
```

```
    homem( T),
    irmaos( T, P),
    progenitor( P, S).

tia ( T, S) :-
    mulher( T),
    irmaos( T, P),
    progenitor( P, S).

primo( A, B) :-
    homem( A),
    progenitor( X, A),
    irmaos( X, Y),
    progenitor( Y, B).


prima( A, B) :-
    mulher( A),
    progenitor( X, A),
    irmaos( X, Y),
    progenitor( Y, B).

avo( A, B) :- pai( A, X), progenitor( X, B).
avoh( A, B) :- mae( A, X), progenitor( X, B).
```

Uso:

SWI

```
?- tio(Tio,Quem). % Quem é tio de quem?
Tio = herbert,
Quem = bart ;
Tio = herbert,
Quem = lisa ;
Tio = herbert,
Quem = maggie ;
false.

?- tia(patty,Quem). % De quem Patty é tia?
Quem = bart ;
Quem = lisa ;
Quem = maggie ;
Quem = ling ;
Quem = bart ;
```

```
Quem = lisa ;
Quem = maggie ;
Quem = ling ;
false.

?- primo(bart,Quem). % De quem Bart é primo?
Quem = ling ;
Quem = ling ;
false.

?- prima(A,B). % Quem é primo de quem?
A = lisa,
B = ling ;
A = lisa,
B = ling ;
A = maggie,
B = ling ;
A = maggie,
B = ling ;
A = ling,
B = bart ;
A = ling,
B = lisa ;
A = ling,
B = maggie ;
A = ling,
B = bart ;
A = ling,
B = lisa ;
A = ling,
B = maggie ;
false.

?- avo(Avo,Quem). % Quem é avô de quem?
Avo = abraham,
Quem = bart ;
Avo = abraham,
Quem = lisa ;
Avo = abraham,
Quem = maggie ;
Avo = clancy,
Quem = bart ;
Avo = clancy,
Quem = lisa ;
Avo = clancy,
```

```
Quem = maggie ;
Avo = clancy,
Quem = ling ;
false.

?- avoh(Avoh,Quem). % Quem é avó de quem?
Avoh = mona,
Quem = bart ;
Avoh = mona,
Quem = lisa ;
Avoh = mona,
Quem = maggie ;
Avoh = jacqueline,
Quem = bart ;
Avoh = jacqueline,
Quem = lisa ;
Avoh = jacqueline,
Quem = maggie ;
Avoh = jacqueline,
Quem = ling ;
false.
```

Comentário

1.2 Desafios lógicos

* **Ex. 4 — (Problema da banda)** Resolver o seguinte problema:

> 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.

Solução:

```
solucao_musicos( S) :-
    banda_solista( S),
    primeiro( X, S),
    toca( X, piano),
```

```
    tocaAntes( Y, Z, S),
    nome( Y, john), toca( Y, sax),
    pais( Z, australia ),
    tocaAntes( Y1, Z1, S),
    nome( Y1, mark), pais( Y1, eua),
    toca( Z1, violino ),
    membros( U, S), pais( U, japao),
    membros( V, S), nome( V, sam).

banda_solista( banda( solista( N1, P1, I1),
                      solista ( N2, P2, I2),
                      solista ( N3, P3, I3 ))).

nome( solista( N,_,_), N).

pais( solista (_, P,_), P).

toca( solista (_,_, I ), I ).

primeiro( X, banda( X,_,_)).

tocaAntes( X, Y, banda( X, Y, Z)).
tocaAntes( X, Z, banda( X, Y, Z)).
tocaAntes( Y, Z, banda( X, Y, Z)).

membros( X, banda( X, Y, Z)).
membros( Y, banda( X, Y, Z)).
membros( Z, banda( X, Y, Z)).
```

Uso:

SWI

```
?- solucao_musicos(Sol).
Sol = banda(solista(mark, eua, piano),
        solista(john, japao, sax),
        solista(sam, australia, violino)) .
```

Comentário

Podemos reescrever o problema dado de uma forma mais lógica:

```
S é uma solução para o problema dos músicos se
  S é uma banda com três solistas,
```

```
  na qual o primeiro membro X de S toca piano, e
  existem dois membros ordenados Y, Z de S tal que
    o nome de Y é John, Y toca saxofone
    e Z vem da Austrália, e
  existem dois membros ordenados Y1, Z1 de S tal que
    o nome de Y1 é Mark, Y1 vem dos Estados Unidos,
    e Z1 toca violino, e
existe um membro U de S que vem do Japão, e
existe um membro V de S cujo nome é Sam.
```

Agora precisamos criar uma forma para a solução. Cada solista possui um nome, um país de origem e um instrumento. Podemos respresentá-lo usando o termo solista (N, P, I), no qual:

N: é o nome do solista;

P: é o país de origem; e

I: é o instrumento que o solista toca.

Com este termo em mãos, a solução para o problema será representada pelo termo
banda(solista(N1, P1, I1), solista (N2, P2, I2), solista (N3, P3, I3))

Ex. 5 — (Problema do jantar) Num fim de semana chuvoso (sexta, sábado e domingo), três mulheres tiveram a mesma ideia: fazer uma sopa para o jantar da família. As sopas foram de agrião, cebola e ervilha. Uma das mulheres se chama Nádia. Conforme as informações a seguir, descubra seus nomes, em que dia da semana cada uma fez sopa e que tipo de sopa preparou para o jantar:

1. Ana Lúcia fez sopa de ervilha.
2. A sopa de cebola foi preparada para o jantar de domingo.
3. Dalva resolveu fazer sopa na sexta-feira.

Solução:

```
solucao_jantar( S):-
    solucao( S),
    membros( X, S),
    nome( X, ana), sopa( X, ervilha),
    membros( Y, S),
    dia( Y, domingo), sopa( Y, cebola),
```

```
    membros( Z, S),
    nome( Z, dalva), dia( Z, sexta ),
    membros( T, S), sopa( T, agriao),
    membros( U, S), nome( U, nadia),
    membros( V, S), dia( V, sexta),
    membros( W, S), dia( W, sabado).

solucao( jantar( mulher( N1, sexta, S1),
                 mulher( N2, sabado, S2),
                 mulher( N3, domingo, S3))).

nome( mulher( N,_,_), N).
dia( mulher(_, D,_), D).
sopa( mulher(_,_, S), S).

membros( M, jantar( M,_,_)).
membros( M, jantar(_, M,_)).
membros( M, jantar(_,_, M)).
```

Uso:

SWI

```
?- solucao_jantar(Sol).
Sol = jantar(mulher(dalva, sexta, agriao),
        mulher(ana, sabado, ervilha),
        mulher(nadia, domingo, cebola)) .
```

Ex. 6 — (Noite de sonhos) Depois de assistirem a vários filmes de aventura num final de semana, Juliano e seus dois irmãos sonharam com aquele mundo de fantasias. Um dos irmãos se chama Nelson. Um dos locais é uma floresta e uma das entidades é um dragão. Com base nas dicas, descubra o nome dos garotos, a entidade que cada um deles encontrou no sonho e o local onde estavam.

1. Roberto sonhou que encontrava um unicórnio.
2. Um dos garotos sonhou que encontrava um bruxo num pântano.
3. Juliano sonhou que estava num castelo.

Solução:

```
solucao_aventura( S):-
    garotos( S),
```

```
    membros( X, S), nome( X, nelson),
    membros( Y, S), local( Y, floresta ),
    membros( Z, S), entidade( Z, dragao),
    membros( T, S), nome( T, roberto), entidade( T, unicornio),
    membros( U, S), entidade( U, bruxo), local( U, pantano),
    membros( V, S), nome( V, juliano), local( V, castelo).

garotos( aventura( garoto( N1, E1, L1), garoto( N2, E2, L2), garoto( N3, E3,
L3))).

membros( X, aventura( X,_,_)).
membros( X, aventura(_, X,_)).
membros( X, aventura(_,_, X)).

nome( garoto( N,_,_), N).
entidade( garoto(_, E,_), E).
local ( garoto(_,_, L), L ).
```

Uso:

SWI

```
solucao_aventura(Sol).
Sol = aventura(garoto(nelson, bruxo, pantano),
               garoto(roberto, unicornio, floresta),
               garoto(juliano, dragao, castelo)) .
```

** **Ex. 7 — (Barcos pesqueiros)** Assim que escurece, cinco barcos pesqueiros saem do porto em busca da pescaria. Com as informações que seguem, descubra o nome e o número de registro de cada barco, o nome de seus mestres arrais e o que foram pescar.

1. O E34 pescará sardinhas.
2. O "Marina" é o E18, enquanto o "Gaivota" navega sob o comando do mestre Benedito.
3. A tripulação do "Jangada" está pegando siris. O "Rei dos Mares" não está a procura de badejos e seu mestre não é Wanderson.
4. Nem a tripulação do "Netuno" nem o mestre Bernardo e sua tripulação pegarão lagostas.
5. O barco do mestre Hélio, E60, não é o "Jangada" nem o "Rei dos Mares".

6. O mestre Fernando espera pescar muitos dourados em um barco cujo nome começa com "M".
7. O registro do "Gaivota" não é E56.
8. Dois registros são E42 e E60.

Solução:

```
resolve( S):-
    solucao( S),
    membros( A, S), registro( A, e34), pesca( A, sardinhas),
    membros( B, S), nome( B, marina), registro( B, e18),
    membros( C, S), nome( C, gaivota), mestre( C, benedito),
    membros( D, S), nome( D, jangada), pesca( D, siris),
    membros( E, S), nome( E, rei_dos_mares),
    membros( F, S), nome( F, netuno),
    membros( G, S), mestre( G, bernardo),
    membros( H, S), mestre( H, helio), registro( H, e60),
    membros( I, S), mestre( I, fernando), pesca( I, dourados),
                    nome( I, NI), comeca_com( NI, m),
    membros( J, S), nome( J, gaivota),
    membros( K, S), registro( K, e42),
    membros( L, S), registro( L, e60),
    membros( M, S), mestre( M, wanderson),
    membros( N, S), pesca( N, lagostas),
    membros( O, S), pesca( O, badejos),
    membros( P, S), registro( P, e56),
    pesca( E, PE), PE \== badejos,
    mestre( E, ME), ME \== wanderson,
    pesca( F, PF), PF \== lagostas,
    pesca( G, PG), PG \== lagostas,
    nome( H, NH), NH \== jangada, NH \== rei_dos_mares,
    registro( J, RJ), J \== e56.


solucao( pescaria( barco( N1, R1, M1, P1),
                   barco( N2, R2, M2, P2),
                   barco( N3, R3, M3, P3),
                   barco( N4, R4, M4, P4),
                   barco( N5, R5, M5, P5))).
```

```
nome( barco( N,_,_,_), N).
registro( barco(_, R,_,_), R).
mestre( barco(_,_, M,_), M).
pesca( barco(_,_,_, P), P).

membros( X, pescaria( X,_,_,_,_)).
membros( X, pescaria(_, X,_,_,_)).
membros( X, pescaria(_,_, X,_,_)).
membros( X, pescaria(_,_,_, X,_)).
membros( X, pescaria(_,_,_,_, X)).

comeca_com( N, L):- atom_chars( N,[ L|_]).
```

Uso:

SWI

```
?- resolve(S).
S = pescaria(barco(rei_dos_mares, e34, bernardo, sardinhas),
             barco(marina, e18, fernando, dourados),
             barco(gaivota, e42, benedito, lagostas),
             barco(jangada, e56, wanderson, siris),
             barco(netuno, e60, helio, badejos)) .
```

Comentário

Neste exercício fizemos uso do predicado pré-definido `atom_chars/2` que converte um átomo em uma lista de caracteres que o integram.

** **Ex. 8 — (Tê Vês)** A última novidade na programação de TV infantil são os Tê Vês, cinco criaturinhas que vivem numa casa mágica em um planeta distante. Com as informações que se seguem, descubra seus nomes, suas cores, o que está usando e o que faz na casa.

1. Luna é amarelo, mas Didi não é verde nem rosa.
2. O Tê Vê verde não usa avental, que é o acessório do cozinheiro, nem é o responsável pelo jardim.
3. O Tê Vê que usa chapéu não é rosa, não é o motorista dos outros nem lava toda a roupa.
4. O Tê Vê lilás é o que lava a roupa mas não usa calção.
5. Dadá faz a faxina da casa, mas não usa chapéu.
6. Bica sempre está de suéter, enquanto o Tê Vê vermelho está sempre de casaco.

7.Um dos Tê Vês se chama Gegê.

Solução:

```
resolve( S):-
    solucao( S),
    membros( A, S), nome( A, luna), cor( A, amarelo),
    membros( B, S), nome( B, didi), cor( B, CB),
    membros( C, S), cor( C, verde), roupa( C, RC),
                    funcao( C, FC),
    membros( D, S), funcao( D, cozinheiro), roupa( D, avental),
    membros( E, S), roupa( E, chapeu), cor( E, CE),
                    funcao( E, FE),
    membros( F, S), cor( F, lilas ),  funcao( F, lavador),
                    roupa( F, RF),
    membros( G, S), nome( G, dada), funcao( G, faxineiro),
                    roupa( G, RG),
    membros( H, S), nome( H, bica), roupa( H, sueter),
    membros( I, S), cor( I, vermelho), roupa( I, casaco),
    membros( J, S), nome( J, gege),
    membros( K, S), cor( K, rosa),
    membros( L, S), funcao( L, jardineiro),
    membros( M, S), funcao( M, motorista),
    membros( N, S), roupa( N, calcao),
    CB \== verde, CB \== rosa,
    RC \== avental,
    FC \== jardineiro,
    CE \== rosa,
    FE \== motorista, FE \== lavador,
    RF \== calcao,
    RG \== chapeu.


solucao( criaturas( tv( N1, C1, R1, F1),
                    tv( N2, C2, R2, F2),
                    tv( N3, C3, R3, F3),
                    tv( N4, C4, R4, F4),
                    tv( N5, C5, R5, F5))).

nome( tv( N,_,_,_), N).
```

```
cor( tv(_, C,_,_), C).
roupa( tv(_,_, R,_), R).
funcao( tv(_,_,_, F), F).

membros( X, criaturas( X,_,_,_,_)).
membros( X, criaturas(_, X,_,_,_)).
membros( X, criaturas(_,_, X,_,_)).
membros( X, criaturas(_,_,_, X,_)).
membros( X, criaturas(_,_,_,_, X)).
```

Uso:

SWI

```
?- resolve(S).
S = criaturas(tv(luna, amarelo, chapeu, jardineiro),
              tv(didi, vermelho, casaco, motorista),
              tv(dada, verde, calcao, faxineiro),
              tv(gege, rosa, avental, cozinheiro),
              tv(bica, lilas, sueter, lavador)) .
```

Comentário

Note que todos os predicados com negação foram colocados no final do programa, depois que todas as variáveis já foram instanciadas.

*** **Ex. 9 — (Fazendeiros e suas fazendas)** Cinco fazendeiros de uma região resolveram trocar os rebanhos tradicionais por criações exóticas. Com as dicas que seguem, diga o nome de cada fazendeiro, quando ele comprou a sua propriedade, sua medida em acres e o animal exótico que cria.

1. Josué entrou para vida de fazenda mais tarde que o homem que agora cria jacarés, cuja fazenda é menor que a de Josué.
2. A fazenda com 3250 acres foi comprada em 1980.
3. As lhamas são criadas na fazenda comprada logo a seguir à maior fazenda.
4. 2500 acres é a medida da fazenda onde são criados javalis, que não foi adquirida em 1956.
5. Yago cuida de veados; ele fez sua estréia no mundo das fazendas em uma ano posterior ao que hoje é criador de avestruzes.
6. Tadeu é o dono da fazenda com 3500 acres (que não cria avestruzes).

7. 1963 foi o ano em que Mariano comprou sua fazenda; sua propriedade é maior que a de Damião, mas não é tão grande quanto a que foi comprada em 1949.
8. Uma das fazendas foi comprada em 1972.
9. Duas fazendas possuem, respectivamente, 2750 e 3600 acres.

Solução:

```
resolve( S):-
  solucao( S),
  membros( A, S), nome( A, josue), data( A, DA), medida( A, MA),
  membros( B, S), animal( B, jacare), data( B, DB), medida( B, MB),
  membros( C, S), medida( C,3250), data( C,1980),
  membros( D, S), animal( D, lhama), data( D, DD), medida( D, MD),
  membros( E, S), data( E, DE), medida( E, ME),
  membros( F, S), medida( F,2500), animal( F, javali), data( F, DF),
  membros( G, S), nome( G, yago), animal( G, veado), data( G, DG),
  membros( H, S), animal( H, avestruz), data( H, DH),
  membros( I, S), nome( I, tadeu), medida( I,3500), animal( I, AI),
  membros( J, S), nome( J, mariano), data( J,1963), medida( J, MJ),
  membros( K, S), nome( K, damiao), medida( K, MK), data( K, DK),
  membros( L, S), data( L,1949), medida( L, ML),
  membros( M, S), data( M,1972),
  membros( N, S), medida( N,2750),
  membros( O, S), medida( O,3600),
  membros( P, S), data( P,1956),
  DA > DB, MA > MB,
  DD > DE, ME> MD,
  DF =\= 1956,
  DG > DH,
  AI \== avestruz,
  DK =\= 1949,
  MJ > MK, MJ < ML.


solucao( fazendas( fazendeiro( N1, D1, M1, A1),
                   fazendeiro( N2, D2, M2, A2),
                   fazendeiro( N3, D3, M3, A3),
                   fazendeiro( N4, D4, M4, A4),
                   fazendeiro( N5, D5, M5, A5))).
```

```
nome( fazendeiro( N,_,_,_), N).
data( fazendeiro(_, D,_,_), D).
medida( fazendeiro(_,_, M,_), M).
animal( fazendeiro(_,_,_, A), A).

membros( X, fazendas( X,_,_,_,_)).
membros( X, fazendas(_, X,_,_,_)).
membros( X, fazendas(_,_, X,_,_)).
membros( X, fazendas(_,_,_, X,_)).
membros( X, fazendas(_,_,_,_, X)).
```

Uso:

SWI

```
?- resolve(Sol).
Sol = fazendas(fazendeiro(josue, 1956, 3600, avestruz),
        fazendeiro(tadeu, 1949, 3500, jacare),
        fazendeiro(yago, 1980, 3250, veado),
        fazendeiro(mariano, 1963, 2750, lhama),
        fazendeiro(damiao, 1972, 2500, javali)) .
```

Comentário

Fizemos uso do predicado =\= que é verdadeiro sempre que seus argumentos *não* são os mesmos números.

1.3 Recursão

Ex. 10 — (Labirinto)

Imagine que a base de conhecimentos seguinte descreva um labirinto. Os fatos determinam quais pontos estão conectados, ou seja, os pontos que se pode alcançar com exatamente um passo. Além disto, imagine que todos os caminhos são ruas de mão única, tal que você somente pode caminhar por elas em uma direção. Assim, você poderá ir do ponto 1 para o ponto 2, mas não poderá ir de 2 para 1.

```
conectado(1,2).
conectado(3,4).
conectado(5,6).
conectado(7,8).
conectado(9,10).
conectado(12,13).
```

```
conectado(13,14).
conectado(15,16).
conectado(17,18).
conectado(19,20).
conectado(4,1).
conectado(6,3).
conectado(4,7).
conectado(6,11).
conectado(14,9).
conectado(11,15).
conectado(16,12).
conectado(14,17).
conectado(16,19).
```

Escreva um predicado `caminho/2` que diga até onde se pode chegar partindo de um determinado ponto, seguindo as conexões na base de conhecimento. Pode-se ir do ponto 5 para o ponto 10? Em quais outros pontos se pode chegar partindo do ponto 1? E quais pontos podem ser alcançados saindo do ponto 13?

Solução:

```
caminho( X, Y) :-
   conectado( X, Y).    % conexão direta
caminho( X, Y) :-
   conectado( X, Z),    % conexão indireta
   caminho( Z, Y).
```

Uso:

SWI

```
?- caminho(5,10).
true .

?- caminho(1,Y).
Y = 2 ;
false.

?- caminho(13,Y).
Y = 14 ;
Y = 9 ;
Y = 17 ;
Y = 10 ;
Y = 18 ;
```

```
false.
```

Ex. 11 — (Viagem pelo mundo) Imagine que você resolva passear mundo afora e possua a seguinte base de conhecimento sobre opções de transporte entre cidades:

```
deCarro( auckland, hamilton).
deCarro( hamilton, raglan).
deCarro( valmont, saarbruecken).
deCarro( valmont, metz).

deTrem( metz, frankfurt).
deTrem( saarbruecken, frankfurt).
deTrem( metz, paris).
deTrem( saarbruecken, paris).

deAviao( frankfurt, bangkok).
deAviao( frankfurt, singapore).
deAviao( paris, losAngeles).
deAviao( bangkok, auckland).
deAviao( losAngeles, auckland).
```

Escreva um predicado `viagem/2` que determine se é possível viajar de um lugar a outro usando qualquer meio de transporte disponível: carro, trem e avião. Por exemplo, seu programa deveria responder `true` para a consulta viagem(valmont, raglan).

Solução:

```
viagem( X, Y):-
   deCarro( X, Y);
   deTrem( X, Y);
   deAviao( X, Y).
viagem( X, Y):-
  ( deCarro( X, Z);
    deTrem( X, Z);
    deAviao( X, Z)),
    viagem( Z, Y).
```

Uso:

SWI

```
?- viagem(valmont,raglan).
true .

?- viagem(auckland,metz).
false.
```

✎ **Comentário**

Ex. 12 — (Viagem pelo mundo – parte II) Usando o predicado `viagem/2` para consultar a base de dados anterior pode-se deduzir que é possível ir de Valmont para Raglan. No caso de estar planejando uma viagem, isto é uma boa informação, mas o que você realmente deseja saber é *como* exatamente ir de Valmont a Raglan. Escreva um predicado `viagem/3` que diga a você como viajar de um lugar para o outro.

Solução:

```
viagem( X, Y, vai( X, Y)) :-
   deCarro( X, Y);
   deTrem( X, Y);
   deAviao( X, Y).
viagem( X, Y, vai( X, Z, C)) :-
  ( deCarro( X, Z);
    deTrem( X, Z);
    deAviao( X, Z)),
    viagem( Z, Y, C).
```

Uso:

SWI

```
?- viagem(valmont,paris,vai(valmont,metz,vai(metz,paris))).
true .

?- viagem(valmont,losAngeles,Caminho).
Caminho = vai(valmont, saarbruecken,
          vai(saarbruecken, paris,
             vai(paris, losAngeles))) ;

Caminho = vai(valmont, metz,
           vai(metz, paris,
              vai(paris, losAngeles))) ;
false.
```

Ex. 13 — (Viagem pelo mundo – parte III) Estenda o predicado `viagem/3` tal que ele não somente diga a você em quais cidades intermediárias você passará, mas também *como* ir de uma cidade a outra, ou seja, de carro, trem ou avião.

Solução:

```
viagem( X, Y, vai( X, de_carro, Y)) :-
  deCarro( X, Y).
viagem( X, Y, vai( X, de_trem, Y)) :-
  deTrem( X, Y).
viagem( X, Y, vai( X, de_aviao, Y)) :-
  deAviao( X, Y).
viagem( X, Y, vai( X, de_carro, Z, C)) :-
  deCarro( X, Z),
  viagem( Z, Y, C).
viagem( X, Y, vai( X, de_trem, Z, C)) :-
  deTrem( X, Z),
  viagem( Z, Y, C).
viagem( X, Y, vai( X, de_aviao, Z, C)) :-
  deAviao( X, Z),
  viagem( Z, Y, C).
```

Uso:

SWI

```
?- viagem(valmont,losAngeles,X).
X = vai(valmont, de_carro, saarbruecken,
      vai(saarbruecken, de_trem, paris,
        vai(paris, de_aviao, losAngeles))) ;

X = vai(valmont, de_carro, metz,
      vai(metz, de_trem, paris,
        vai(paris, de_aviao, losAngeles))) ;
false.
```

Ex. 14 — ()

Solução:

Uso:

SWI

✎ **Comentário**

Ex. 15 — ()

Solução:

Uso: SWI

✎ **Comentário**

Ex. 16 — ()

Solução:

Uso: SWI

✎ **Comentário**

Ex. 17 — ()

Solução:

Uso: SWI

✎ **Comentário**

2 Exercícios Diversos

2.1 Teoria dos Números

Ex. 18 — (Máximo Divisor Comum) Qual é o máximo divisor comum entre dois inteiros não-negativos?

Solução:

```
mdc(0, Y, Y) :- !.
mdc( X, Y, MDC) :-
    X < Y, !,
    X1 is Y mod X,
    mdd( X1, X, MDC).
mdc( X, Y, MDC) :-
    X1 is X mod Y,
    mdd( X1, Y, MDC),!.
```

Uso:

SWI

```
?- mdc(420,66,MDC).    % mdc(+,+,-)
MDC = 6.
?- mdc(420,66,6).    % mdc(+,+,+)
true.
```

Comentário

O predicado `mdc/3` implementa o algoritmo de Euclides. Trata-se de um método simples e eficiente de encontrar o máximo divisor comum entre dois números inteiros não-negativos. É um dos algoritmos mais antigos, conhecido desde que surgiu nos Livros VII e X da obra *Os Elementos* de Euclides por volta de 300aC. O algoritmo provavelmente não foi concebido por Euclides, que compilou resultados de matemáticos anteriores em *Os Elementos*. O matemático e historiador *Bartel van der Waerden* sugere que o Livro VII provém de um texto em teoria dos números escrito por matemáticos da escola Pitagórica.

Ex. 19 — (Mínimo Múltiplo Comum) xcx Qual é o mínimo múltiplo comum entre dois inteiros não-negativos?

Solução:

```
mmc( X, Y, MMC) :-
    mdc( X, Y, MDC),
    MMC is X* Y/ MDC.
```

Uso:

SWI

```
?- mmc(420,66,MMC).    % mmc(+,+,-)
MDC = 4620.
?- mmc(420,66,4620).    % mmc(+,+,+)
true.
```

Comentário
O predicado `mdc/3` está implementado no exercício 18.

Ex. 20 — (Intervalo (*Range*)) Como criar uma lista monótona crescente que contenha todos os número inteiros entre dois números inteiros dados (em ordem crescente)?

Solução:

```
intervalo( Fim, Fim,[ Fim]).
intervalo( Inicio , Fim,[ Inicio | Resto]) :-
    Inicio1 is Inicio +1,
    intervalo( Inicio1 , Fim, Resto ),!.
```

Uso:

SWI

```
?- intervalo(1,7,X).    %intervalo(+,+,-)
X = [1, 2, 3, 4, 5, 6, 7].
?- intervalo(7,2,[2,3,4,5,6,7]).    %intervalo(+,+,+)
false.
?- intervalo(2,7,[2,3,4,5,6,7]).    %intervalo(+,+,+)
true.
?- intervalo(2,X,[2,3,4,5,6,7]).    %intervalo(+,-,+)
X = 7.
?- intervalo(X,Y,[2,3,4,5,6,7]).    %intervalo(-,-,+)
X = 2,
Y = 7.
```

Comentário
Uma lista é monótona crescente se para cada par dos seus elementos consecutivos a e b, tem-se que $a \leq b$.

Ex. 21 — (Remoção de múltiplos de um número de uma lista) Como remover todos os múltiplos de k de uma lista dada?

Solução:

```
remove_multiplos( K,[ H| T], R) :-
  0 is  H mod  K,!,              % se é múltiplo, elimina-o
   remove_multiplos( K, T, R).
remove_multiplos( K,[ H| T],[ H| R]) :-
   remove_multiplos( K, T, R),!. % caso contrário, mantê-lo
remove_multiplos(_,[ ],[ ]) :- !.
```

Uso:

SWI

```
?- remove_multiplos(2,[2,3,4,5,6,7],X).   %remove_multiplos(+,+,-)
X = [3, 5, 7].
?- remove_multiplos(2,[2,3,4,5,6,7],[3,5,7]).   %remove_multiplos(+,+,+)
true.
```

✎ **Comentário**
Os cortes (*cuts*) *foram introduzidos para tornar o predicado* `removeMultiplos/3` *determinístico e mais eficiente.*

Ex. 22 — (Crivo de Erastóstenes) Quais são os números primos no intervalo entre 1 e um inteiro positivo dado?

Solução:

```
primos( N, ListaPrimos) :-
     N > 1,
     intervalo(2, N, Lista),
     Parada is  int( sqrt( N)),     % parte inteira da raiz quadrada de N
     crivo( Lista, Parada, ListaPrimos),!.

crivo([ H| T], Parada,[ H| T]) :-
     H > Parada,!.
crivo([ H| T], Parada,[ H| R]) :-
     remove_multiplos( H, T, NovoT),
     crivo( NovoT, Parada, R).
```

Uso:

SWI

```
?- primos(25,X).   % primos(+,-)
X = [2, 3, 5, 7, 11, 13, 17, 19, 23].
```

```
?- primos(5,[2,3,5]).    % primos(+,+)
true.
```

Comentário
O Crivo de Eratóstenes é um método algorítmico simples e prático para determinar os números primos até um valor limite. Foi criado pelo matemático grego Eratóstenes (c. 285-194 a.C.), o terceiro bibliotecário-chefe da Biblioteca de Alexandria.
Para exemplificá-lo, vamos determinar a lista de números primos entre 1 e 25. Inicialmente, determina-se o maior número a ser checado. Ele corresponde ao piso da raiz quadrada do valor limite. No caso, a raiz quadrada de 25, que é 5. Cria-se uma lista de todos os números inteiros de 2 (o primeiro primo) até o valor limite (25) : 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25. O primeiro número da lista é um número primo, 2. Então, a partir de 2, remove-se da lista todos os múltiplos do número primo encontrado. No nosso exemplo, a lista fica: 2 3 5 7 9 11 13 15 17 19 21 23 25. O próximo número da lista é primo. Repita o procedimento anterior para a lista após o elemento 2. Dessa forma, o próximo número (3) é primo. Remove-se os múltiplos de 3, a lista fica: 2 3 5 7 11 13 17 19 23 25. O próximo número, 5, também é primo, procedendo analogamente, a lista fica: 2 3 5 7 11 13 17 19 23. Como 5 é o último número a ser verificado, conforme determinado inicialmente, a lista final determinada contém somente números primos entre 1 e 25.

2.2 Listas

Ex. 23 — (Pertinência de um elemento a uma lista) Como determinar se um elemento pertence a uma lista?

Solução:

```
pertence( H,[ H| _ ]).
pertence( X,[_| T]) :-  pertence( X, T).
```

Uso:

SWI

```
?- pertence(2,[1,2,3]).    % pertence(+,+)
true
?- pertence(X,[1,2,3]).    % pertence(-,+)
```

```
X = 1 ;
X = 2 ;
X = 3 ;
false.
```

Comentário
O predicado `pertence/2` já está implementado no SWI-Prolog (biblioteca *lists*), como também nem outras implementações, todavia com o functor `member`. É aconselhável sempre utilizar as bibliotecas disponíveis no SWI-Prolog, nesse caso utilizando `member/2` em programas no dia-a-dia.

Ex. 24 — (Multiplicidade em uma lista) Como verificar se em uma lista existe múltipla ocorrência de algum elemento?

Solução:

```
multiplo([ H| T]) :- pertence( H, T).
multiplo([_| T]) :- multiplo( T).
```

Uso:

SWI

```
?- multiplo([1,2,3]).   % multiplo(+)
false.
?- multiplo([1,2,1]).   % multiplo(+)
true
```

Comentários

- Exemplo de aplicação do predicado `pertence/2` (vide exercício 23);
- De forma geral, em todo documento, não serão acentuados os functores e variáveis nos programas Prolog, nesse exercício o functor *multiplo* que refere-se à palavra *múltiplo* não está acentuado.

Ex. 25 — (Verificação da pertinência de um elemento a uma lista) Como apenas verificar se um elemento pertence a uma lista?

Solução:

```
verifica_pertence( H,[ H| _]) :- !.
verifica_pertence( X,[_| T]) :- verifica_pertence( X, T ),!.
```

Uso:

SWI

```
?- verifica_pertence(2,[1,2,3]).    % verifica_pertence(+,+)
true.
?- verifica_pertence(4,[1,2,3]).    % verifica_pertence(+,+)
false.
?- verifica_pertence(X,[1,2,3]).    % verifica_pertence(-,+) - uso indevido
X = 1.
```

✎ **Comentário**
O predicado `verifica_pertence/2` é determinístico, contrariamente ao predicado `pertence/2` que é não-determinístico.

Ex. 26 — (Seleção de um elemento de uma lista) Como selecionar um elemento de uma lista? Uma vez selecionado esse elemento, quais os elementos que restam na lista original?

Solução:

```
seleciona( H,[ H| T], T).
seleciona( Elem,[ H| T],[ H| R]) :- seleciona( Elem, T, R).
```

Uso:

SWI

```
?- seleciona(X,[1,2,3],R).        % seleciona(-,+,-)
X = 1
R = [2,3] ;
X = 2
R = [1,3] ;
X = 3
R = [1,2] ;
false.
?- seleciona(2,[1,2,3],R).        % seleciona(+,+,-)
R = [1,3] ;
false.
?- seleciona(2,[1,2,3],[1,4]).    % seleciona(+,+,+)
false.
?- seleciona(1,Lista,[2,3]).      % seleciona(+,-,+)
Lista = [1,2,3] ;
Lista = [2,1,3] ;
Lista = [2,3,1] ;
false.
```

✎ **Comentário**
O predicado `seleciona/3` está implementado na biblioteca *lists* no SWI-Prolog, todavia com o functor `select`.

Ex. 27 — (Seleção dos N primeiros elementos de uma lista) Como selecionar N elementos do início de uma lista?

Solução:

```
seleciona_n_primeiros(0,_,[ ]).
seleciona_n_primeiros( Nro,[ H| T],[ H| R]) :-
    NovoNro is Nro-1,
    seleciona_n_primeiros( NovoNro, T, R).
```

Uso:

SWI

```
?- seleciona_n_primeiros(3,[a,b,c,d,e,f],X).   % seleciona_n_primeiros(+,+,-)
X = [a, b, c].
?- seleciona_n_primeiros(2,[a,b,c,d],[a,b]).   % seleciona_n_primeiros(+,+,+)
true.
?- seleciona_n_primeiros(3,X,[a,b,c]).   % seleciona_n_primeiros(+,-,+)
X = [a, b, c|_G2205].
```

Comentário

O predicado `seleciona/3` está implementado na biblioteca *lists* no SWI-Prolog, todavia com o functor `select`.

Ex. 28 — (Problema matemático simples) Seja L uma lista finita de números. Determine uma solução para $x + y = z$, onde x e y pertencem a L, x e y distintos e z é um número dado?

Solução:

```
determina_xy( L, Z, X, Y) :-
    seleciona( X, L, RestoL),
    seleciona( Y, RestoL,_),
    X \= Y,
    Z is X + Y.
```

Uso:

SWI

```
?- determina_xy([1,2,3,4,5,6,7,8],13,X,Y).   % determina_xy(+,+,-,-)
X = 5
Y = 8 ;
X = 6
Y = 7 ;
X = 7
Y = 6 ;
X = 8
```

```
Y = 5 ;
false.
?- determina_xy([1,2,3,4,5,6,7,8],13,6,7).    % determina_xy(+,+,+,+)
true
?- determina_xy([1,2,3,4,5,6,7,8],13,6,X).    % determina_xy(+,+,+,-)
X = 7 ;
false.
?- determina_xy([1,2,3,4,5,6,7,8],13,X,7).    % determina_xy(+,+,-,+)
X = 6 ;
false.
```

Comentário
Exemplo de aplicação do predicado `seleciona/3` (vide exercício 26).

* **Ex. 29 — (Problema Criptográfico)** Sérgio recebeu de seu colega uma senha criptografada da forma "SEND+MORE=MONEY". Para resolver o problema, Sérgio considerou corretamente que a cada letra corresponderia um único dígito decimal (de 0 a 9), deduziu também que "M" é o dígito 1 e "S" deve ser maior que 0. Qual é a senha criptografada?

Solução:

```
resolveProbCripto([ S, E, N, D,+, M, O, R, E,=, M, O, N, E, Y]) :-
    M = 1,
    seleciona( D ,[0,2,3,4,5,6,7,8,9],  Resto1),
    seleciona( E, Resto1, Resto2),
    Y is ( D+ E) mod 10,
    V1 is ( D+ E) // 10,
    seleciona( Y, Resto2, Resto3),
    seleciona( N, Resto3, Resto4),
    seleciona( R, Resto4, Resto5),
    E is ( V1+ N+ R) mod 10,
    V2 is ( V1+ N+ R) // 10,
    seleciona( O, Resto5, Resto6),
    N is ( V2+ E+ O) mod 10,
    V3 is ( V2+ E+ O) // 10,
    seleciona( S, Resto6,_),
    O is ( V3+ S+ M) mod 10,
    M is ( V3+ S+ M) // 10.
```

Uso: SWI

```
?- resolveProbCripto(X).                % resolverProbCripto(+)
X = [9, 5, 6, 7, +, 1, 0, 8, 5|...] ;
false.
?- resolveProbCripto(X), write(X).      % imprimiu-se a resposta
[9,5,6,7,+,1,0,8,5,=,1,0,6,5,2]
X = [9, 5, 6, 7, +, 1, 0, 8, 5|...] ;
false.
```

A senha criptografada é 9567108510652! Trata-se da única solução ao problema, pois, ao tentarmos encontrar mais soluções, o SWI-Prolog retornou `false`.

✎ **Comentário**
Exemplo de aplicação do predicado `seleciona/3` (vide exercício 26).

Ex. 30 — (Eliminação do último elemento de uma lista) Como eliminar apenas o último elemento de uma lista?

Solução:

```
elimina_ultimo([_],[ ]).
elimina_ultimo([ H| T],[ H| R]) :- elimina_ultimo( T, R).
```

Uso:

SWI

```
?- elimina_ultimo([1,2,3],X).   % elimina_ultimo(+)
X = [1,2] ;
false.
?- elimina_ultimo([ ],X).       % elimina_ultimo(+)
false.
```

Ex. 31 — (Determinação do último elemento de uma lista) Qual é o último elemento de uma lista dada?

Solução:

```
ultimo([ X], X).
ultimo([_| T], X) :- ultimo( T, X).
```

Uso:

SWI

```
?- ultimo([1,2,3],X).   % ultimo(+,-)
X = 3 ;
false.
?- ultimo([ ],X).       % ultimo(+,-)
```

```
false.
?- ultimo([1,2,3],2).    % ultimo(+,+)
false.
```

Comentários

- O predicado `ultimo/2` está implementado na biblioteca *lists* no SWI-Prolog, todavia com o functor `last`;
- Você pode sustituir o predicado `ultimo(Lista,Elemento)` pelo predicado `append(_, [Elemento], Lista)` de forma alternativa para alcançar maior portabilidade.

Ex. 32 — (Elemento do meio de uma lista) Qual o elemento do meio de uma lista dada?

Solução:

```
elemento_meio([ X], X).
elemento_meio([_| T], X) :-
    elimina_ultimo( T, PrefixoT),
    elemento_meio( PrefixoT, X).
```

Uso:

SWI

```
?- elemento_meio(X,[1,2,3]).    % elemento_meio(+)
X = 2 ;
false.
?- elemento_meio(X,[1,2,2,1]).  % elemento_meio(+)
false.
```

Comentários

- O predicado `elemento_meio/2` deve falhar se a lista apresentar comprimento par;
- Exemplo de aplicação do predicado `elimina_ultimo/2` (vide exercício 30.

Ex. 33 — (Comprimento de uma lista) Qual é o comprimento de uma lista dada?

Solução:

```
comprimento([ ],0).
comprimento([_| T], C) :-
```

```
    comprimento( T, CT),
    C is  CT + 1.
```

Uso:

SWI

```
?- comprimento([a,b,c],X).    % comprimento(+,-)
X = 3.
?- comprimento([a,b,c],3).    % comprimento(+,+)
true.
?- comprimento(X,2).          % comprimento(-,+)
X = [_G757,_G760]
```

Comentários

- Essa implementação de `comprimento/2` não é eficiente, não há recursão de cauda. Serve apenas para ser utilizada em casos onde o número de elementos na lista é pequeno;
- No exemplo, para a questão "?- comprimento(X,2)." a resposta é uma lista de dois elementos genéricos (variáveis livres). Esse tipo de uso é muito específico e raro, possibilita-se criar uma solução genérica para um problema onde posteriormente calcular-se-á os seus valores. Após a resposta aperte a tecla ESC, pois se você apertar ENTER seu programa entrará em *loop*.
- O predicado `comprimento/2` está implementado no SWI-Prolog, todavia com o functor `length`.

Ex. 34 — (Comprimento de uma lista de forma eficiente) Qual é o comprimento de uma lista dada?

Solução:

```
comprimento2( Lista, C) :-
    comprimento2( Lista,0, C).

comprimento2([ ], Ac, Ac).          %cláusula de parada
comprimento2([_| T], Ac, C) :-
    NAc is  Ac + 1,                 %incremento do acumulador
    comprimento2( T, NAc, C).
```

Uso:

SWI

```
?- comprimento2([a,b,c],X).    % comprimento2(+,-)
X = 3.
```

```
?- comprimento2([a,b,c],3).     % comprimento2(+,+)
true.
?- comprimento2(X,2).           % comprimento2(-,+)
X = [_G757,_G760]
```

✎ **Comentário**
Essa implementação de `comprimento/2` é eficiente, trata-se de uma recursão de cauda.

Ex. 35 — (Concatenação de duas listas) Como concatenar (juntar) duas listas quaisquer?

Solução:

```
concatena([ ], L, L).
concatena([ H| T], L,[ H| R]) :- concatena( T, L, R).
```

Uso:

SWI

```
?- concatena([a,b],[1,2],Lista).        % concatena(+,+,-)
Lista = [a,b,1,2].
?- concatena([a,b],[1,2],[a,b,1,2]).    % concatena(+,+,+)
yes.
?- concatena(X,Y,[a,b,1,2]).            % concatena(-,-,+)
X = [ ]
Y = [a,b,1,2];
X = [a]
Y = [b,1,2];
X = [a,b]
Y = [1,2];
X = [a,b,1]
Y = [2];
X = [a,b,1,2]
Y = [ ];
false.
```

✎ **Comentários**

- O predicado `concatena/3` é não-determinístico. A questão "?- concatena(X,Y,[a,b,1,2])." apresenta 5 respostas corretas;
- O predicado `concatena/3` está implementado na biblioteca *lists* no SWI-Prolog, todavia com o functor append.

Ex. 36 — (Elemento do meio de uma lista (versão 2)) Qual o elemento do meio de uma lista dada?

Solução:

```
elemento_meio( Lista, X) :-
    concatena( Metade1,[ X| Metade2], Lista),
    comprimento( Metade1, Comp),
    comprimento( Metade2, Comp).
```

Uso:

SWI

```
?- elemento_meio(X,[1,2,3]).    % elemento_meio(+)
X = 2 ;
false.
?- elemento_meio(X,[1,2,2,1]).   % elemento_meio(+)
false.
```

Comentários

- O predicado `elemento_meio/2` é ineficiente se comparado à implementação apresentada no exercício 32;
- Exemplo de aplicação dos predicados `comprimento/2` e `concatena/3` (vide exercícios 34 e 35, respectivamente).

Ex. 37 — (Divisão de uma lista ao meio sem alterar ordem dos elementos) Como dividir uma lista em duas metades, considerando que:

1. no caso da lista ter comprimento par, as metades deverão ter o mesmo comprimento;
2. no caso da lista ter comprimento ímpar, as metades devem apresentar comprimentos sucessivos;
3. a ordem dos elementos *não pode ser alterada*.

Solução:

```
divide( Lista , Metade1, Metade2) :-
    concatena( Metade1, Metade2, Lista),
    comprimento( Metade1, Comp1),
    comprimento( Metade2, Comp2),
    abs( Comp1- Comp2) =< 1.
```

Uso:

SWI

```
?- divide([1,2,3,4,5,6],A,B).        % divide(+,-,-)
A = [1, 2, 3],
B = [4, 5, 6] ;
false.
?- divide([1,2,3,4,5,6,7],A,B).    % divide(+,-,-)
A = [1, 2, 3],
B = [4, 5, 6, 7] ;
A = [1, 2, 3, 4],
B = [5, 6, 7] ;
false.
?- divide([ ],X,Y).    % divide(+,-,-)
X = [ ],
Y = [ ] ;
false.
?- divide([1,2,3,4,5,6,7],[1,2,3,4],Y).    % divide(+,+,-)
Y = [5, 6, 7].
?- divide([1,2,3,4,5,6,7],X,[5, 6, 7]).    % divide(+,-,+)
Y = [1,2,3,4] ;
false.
```

Comentários

- Essa abordagem é bela (beleza intrínseca declarativa), todavia ineficiente.
- Exemplo de aplicação dos predicados `comprimento/2` e `concatena/3` (vide exercícios 34 e 35, respectivamente).

Ex. 38 — (Divisão de uma lista ao meio sem alterar a ordem dos elementos (versão 2)) Como dividir uma lista em duas metades, considerando:

1. no caso da lista ter comprimento par, as metades deverão ter o mesmo comprimento;
2. no caso da lista ter comprimento ímpar, as metades devem apresentar comprimentos sucessivos;
3. a ordem dos elementos *não pode ser alterada*.

Solução:

```
divide2( Lista, Metade1, Metade2) :-
    comprimento( Lista, Comp),
    Meio is  Comp // 2,    % divisão inteira
    divide2( Lista, Meio,[ ], Metade1, Metade2).
```

```
divide2( Lista ,0, Metade1, Metade1, Lista).
divide2([ H| T], Meio, Ac, Metade1, Metade2) :-
    NMeio is  Meio-1,
    concatena( Ac,[ H], NAc),
    divide2( T, NMeio, NAc, Metade1, Metade2).
```

Uso:

SWI

```
?- divide2([1,2,3,4,5,6,7],X,Y).    % divide(+,-,-)
X = [1, 2, 3],
Y = [4, 5, 6, 7] ;
false.
?- divide2([1,2,3,4,5,6],X,Y).      % divide(+,-,-)
X = [1, 2, 3],
Y = [4, 5, 6] ;
false.
?- divide2([ ],X,Y).        % divide(+,-,-)
X = [ ],
Y = [ ].
?- divide2([1,2,3,4,5,6,7],[1,2,3],Y).   % divide(+,+,-)
Y = [4, 5, 6, 7] ;
false.
?- divide2([1,2,3,4,5,6,7],[1,2,3,4],Y). % divide(+,+,-)
false.
```

Comentários

- Essa abordagem é mais eficiente que a anterior, todavia tem uso mais restrito, como podemos verificar nas questões "?- divide2([1,2,3,4,5,6],X,Y)." e "?- divide2([1,2,3,4,5,6,7],[1,2,3,4],Y)." .
- Exemplo de aplicação dos predicados `comprimento/2` e `concatena/3` (vide exercícios 34 e 35, respectivamente).

Ex. 39 — (Divisão de uma lista ao meio) Como dividir uma lista em duas metades, considerando:

1. no caso da lista ter comprimento par, as metades deverão ter o mesmo comprimento;
2. no caso da lista ter comprimento ímpar, as metades devem apresentar comprimentos sucessivos;
3. a ordem dos elementos *pode ser alterada*.

Solução:

```
divide3([ ],[ ],[ ]).
divide3([ H],[ H],[ ]).
divide3([ H],[ ],[ H]).
divide3([ H1, H2| T],[ H1| Metade1],[ H2| Metade2]) :-
    divide3( T, Metade1, Metade2).
```

Uso:

SWI

```
?- divide3([ ],X,Y).    % divide(+,-,-)
X = [ ],
Y = [ ].
?- divide3([1,2,3,4,5,6],X,Y).    % divide(+,-,-)
X = [1, 3, 5],
Y = [2, 4, 6].
?- divide3([1,2,3,4,5,6,7],X,Y).    % divide(+,-,-)
X = [1, 3, 5, 7],
Y = [2, 4, 6] ;
X = [1, 3, 5],
Y = [2, 4, 6, 7] ;
false.
?- divide3([1,2,3,4,5,6,7],[1,2,3,4],Y).    % divide(+,+,-)
false.
?- divide3([1,2,3,4,5,6,7],[1,3,5],Y).    % divide(+,+,-)
Y = [2, 4, 6, 7] ;
false.
```

Comentário
Essa abordagem é eficiente e não faz uso do predicado `comprimento/2`.

Ex. 40 — (Divisão de uma lista ao meio (versão 4)) Como dividir uma lista em duas metades, considerando:

1. no caso da lista ter comprimento par, as metades deverão ter o mesmo comprimento;
2. no caso da lista ter comprimento ímpar, as metades devem apresentar comprimentos sucessivos;
3. a ordem dos elementos *pode ser alterada*.

Solução:

```
divide4([ ],[ ],[ ]).
divide4([ H| T],[ H| Metade1], Metade2) :-
```

```
    divide4( T, Metade2, Metade1).
```

Uso:

SWI

```
?- divide4([ ],X,Y).    % divide(+,-,-)
X = [ ],
Y = [ ].
?- divide4([1,2,3,4,5,6],X,Y).    % divide(+,-,-)
X = [1, 3, 5],
Y = [2, 4, 6].
?- divide4([1,2,3,4,5,6,7],X,Y).    % divide(+,-,-)
X = [1, 3, 5, 7],
Y = [2, 4, 6].
?- divide4([1,2,3,4,5,6,7],[1,2,3,4],Y).    % divide(+,+,-)
false.
?- divide4([1,2,3,4,5,6,7],[1,3,5,7],Y).    % divide(+,+,-)
Y = [2, 4, 6].
?- divide4([1,2,3,4,5,6,7],X,[2,4,6]).    % divide(+,-,+)
X = [1, 3, 5, 7].
```

✎ **Comentário**

Essa abordagem é tão eficiente quanto ao exercício 39 e também não faz uso do predicado `comprimento/2`.

Ex. 41 — (Dividir uma lista em uma posição) Como dividir uma lista dada em uma posição determinada? Separe a lista em duas listas, contendo a primeira os N primeiros elementos da lista original e a segunda, os restantes elementos.

Solução:

```
divideN([ H| T], N,[ H| R], Resto) :-
   N > 0,
   N1 is  N-1,
   divideN( T, N1, R, Resto).
divideN( Lista,0,[ ], Lista ),!.
```

Uso:

SWI

```
?- divideN([5,4,1,3,2,6], 3, ListaN, ListaR).
ListaN = [5, 4, 1],
ListaR = [3, 2, 6].
?- divideN([5,4,1,3,2,6], 4, ListaN, ListaR).
ListaN = [5, 4, 1, 3],
ListaR = [2, 6].
```

Ex. 42 — (Inversão de uma lista) Como inverter uma lista dada?

Solução:

```
inverte([ ],[ ]).
inverte([ X| Xs], ListaInv) :-
    inverte( Xs, ListaXsInv),
    concatena( ListaXsInv,[ X], ListaInv).
```

Uso:

SWI

```
?- inverte([a,b,c],X).        % inverso(+,-)
X = [c,b,a].
?- inverte([b,c],[c,a]).      % inverso(+,+)
false.
?- inverte(X,[c,b,a]).        % inverso(-,+) - uso restrito
X = [a,b,c]
```

Comentários

- Essa implementação do predicado `inverte/2` não é eficiente, não há recursão de cauda. Serve apenas para ser utilizada em casos onde o número de elementos na lista é pequeno;
- Nesse exemplo, para a questão "?- inverte(X,[c,b,a])." temos como resposta "X = [a,b,c]". Todavia, se você pressionar ENTER ou ";", o programa entrará em *loop*. Assim, após a resposta você deve pressionar ESC;
- O predicado `inverte/2` está implementado na biblioteca *lists* no SWI-Prolog, todavia com o functor `reverse`. Essa implementação usa "listas diferença" e será apresentada na seção reservada a esse conceito.
- Exemplo de aplicação do predicado `concatena/3` (vide exercício 35).

Ex. 43 — (Inversão de uma lista de forma eficiente) Como inverter uma lista dada?

Solução:

```
inverte2( Lista , ListaInv) :- inverte2( Lista ,[ ], ListaInv ).

inverte2([ ], Ac, Ac).
inverte2([ H| T], Ac, ListaInv) :- inverte2( T,[ H| Ac], ListaInv).
```

Uso:

SWI

```
?- inverte2([a,b,c],X).        % inverso(+,-)
X = [c,b,a].
?- inverte2([b,c],[c,a]).      % inverso(+,+)
false.
?- inverte2(X,[c,b,a]).        % inverso(-,+) - uso sem restrições
X = [a,b,c].
```

Comentários

- Essa implementação de `inverte/2` é mais eficiente que a apresentada no exerício 42, há recursão de cauda.
- No exemplo, na questão "?- inverte(X,[c,b,a])." temos como resposta única "X = [a,b,c]", determinística.

Ex. 44 — (Prefixo de uma lista ou *string*) Quais são os prefixos de uma lista (ou *string*)?

Solução:

```
prefixo([ ], _).
prefixo([ H| T], [ H| R]) :-
    prefixo( T, R).
```

Uso:

SWI

```
?- prefixo(X,[1,2,3]).    % prefixo(-,+)
X = [ ] ;
X = [1] ;
X = [1, 2] ;
X = [1, 2, 3] ;
false.
?- prefixo(X,[ ]).    % prefixo(-,+)
X = [ ] ;
false.
?- prefixo(X,"amor").    % prefixo(-,+)
X = [ ] ;
X = [97] ;                % letra a
X = [97, 109] ;           % letras a, m
X = [97, 109, 111] ;      % letras a, m, o
X = [97, 109, 111, 114] ;    % letras a, m, o, r
false.
?- prefixo("am","amor").    % prefixo(+,+)
true.
```

Comentário
O predicado `prefixo/2` está implementado na biblioteca *lists* no SWI-Prolog, todavia com o functor `prefix`.

Ex. 45 — (Sufixo de uma lista ou *string*) Quais são os sufixos de uma lista (ou *string*)?

Solução:

```
sufixo( X, X).
sufixo( X,[_| T]):- sufixo( X, T).
```

Uso:

SWI

```
?- sufixo(X,[1,2,3]).    % sufixo(-,+)
X = [1, 2, 3] ;
X = [2, 3] ;
X = [3] ;
X = [ ] ;
false.
?- sufixo([2,3],[1,2,3]).    % sufixo(+,+)
true
.
?- sufixo(X,"amor").    % sufixo(-,+)
X = [97, 109, 111, 114] ;
X = [109, 111, 114] ;
X = [111, 114] ;
X = [114] ;
X = [ ] ;
false.
?- sufixo("or","amor").    % sufixo(+,+)
true
.
```

Ex. 46 — (Particionar uma lista por um pivô) Como ordenar uma lista em ordem ascendente?

Solução:

```
particiona(_, [ ], [ ], [ ]) :- !.
particiona( Pivo, [ H| T], [ H| Menores], Maiores) :-
    H @< Pivo, % @ para strings
    !,
```

```
    particiona( Pivo, T, Menores, Maiores).
particiona( Pivo, [ H| T], Menores, [ H| Maiores]) :-
    particiona( Pivo, T, Menores, Maiores).
```

Uso:

SWI

```
?- particiona(4,[3,8,9,0,5,2],Men,Mai).   % particiona(+,+,-,-)
Men = [3, 0, 2],
Mai = [8, 9, 5].
?- particiona(4,[3,8,9,0,5,2],[3,0,2],Mai).   % particiona(+,+,+,-)
Mai = [8, 9, 5].
?- particiona(4,[3,8,9,0,5,2],Men,[8,9,5]).   % particiona(+,+,-,+)
Mai = [3, 0, 2].
?- particiona(4,[3,8,9,0,5,2],[3,0,2],[8,9,5]).   % particiona(+,+,+,+)
true.
?- particiona("mo",["ad","mr","ol","bc","de","po"],Men,Mai). % (strings)
Men = [[97, 100], [98, 99], [100, 101]],
Mai = [[109, 114], [111, 108], [112, 111]].
```

Ex. 47 — (Ordenação *Quicksort*) Como ordenar uma lista em ordem ascendente?

Solução:

```
quicksort([ ],[ ]) :- !.
quicksort([ Pivo| T], ListaOrd) :-
    particiona( Pivo, T, ListaMenores, ListaMaiores),
    quicksort( ListaMenores, MenoresOrd),
    quicksort( ListaMaiores, MaioresOrd),
    concatena( MenoresOrd, [ H| MaioresOrd], ListaOrd).
```

Uso:

SWI

```
?- quicksort([1,0,4,3,8,5,6,2,7],X).   % quicksort(+,-)
X = [0, 1, 2, 3, 4, 5, 6, 7, 8].
?- quicksort([1,0,4,3],[0,1,3,4]).   % quicksort(+,+)
true.
```

Comentários

- São utilizados os predicados `particiona/4` e `concatena/2` (respectivamente, vide exercícios 35 e 46).
- Para ordenar a lista em ordem decrescente basta alterar o predicado `particiona`, trocando < por >.

Ex. 48 — (Ordenação *Quicksort* (versão DCG)) Como ordenar uma lista em ordem ascendente?

Solução:

```
quicksortDCG( Lista, ListaOrd) :-
    quicksortDCG( Lista, ListaOrd,[ ]).

quicksortDCG([ ])      --> [ ].
quicksortDCG([ Pivo| T]) -->
    { particiona( Pivo, T, Menores, Maiores) },
    quicksortDCG( Menores), [ Pivo], quicksortDCG( Maiores).
```

Uso:

SWI

```
?- quicksortDCG([1,0,4,3,5,6,2,7],X).   % quicksortDCG(+,-)
X = [0, 1, 2, 3, 4, 5, 6, 7].
?- quicksortDCG([1,0,4,3],[0,1,3,4]).   % quicksortDCG(+,+)
true.
```

Comentário
É utilizado o predicado `particiona/4` (vide exercício 46).

Ex. 49 — (Sequência de Fibonacci) Qual o n-ésimo termo na sequência de Fibonacci (0,1,1,2,3,5,...)?

Solução:

```
fib (1,0).
fib (2,1).
fib( N, F) :-
  N > 2,
  N1 is  N-1,
  N2 is  N-2,
  fib( N1, F1),
  fib( N2, F2),
  F is  F1+ F2,!.
```

Uso:

SWI

```
?- fib(6,F).   % fib(+,-)
F = 5.
?- fib(6,5).   % fib(+,+)
true.
```

✎ **Comentário**
O código acima é puramente declarativo, dessa forma elegante. Todavia é muito ineficiente, pois há a repetição do cálculo dos termos predecessores na sequência várias vezes. Dado um número inteiro positivo N, o número de chamadas do predicado `fib/` para o cálculo do N-ésimo número na sequência de Fibonacci ($C(N)$) é dado por:

$$C(1) = 1; C(2) = 1; c(N) = C(N-1) + C(N-2) + 1, \text{se} N > 2.$$

Dessa forma, para o cálculo de `fib(40,F)`, o Prolog fará 204.668.309 chamadas do predicado `fib/2`.

Ex. 50 — (Sequência de Fibonacci (versão 2)) Qual o n-ésimo termo na sequência de Fibonacci (0,1,1,2,3,5,...)?

Solução:

```
fibonacci( N, F) :-
    fibonacci( N,_, F).

fibonacci(1,_,0).
fibonacci (2,0,1).
fibonacci( N, F1, F) :-
        N > 2,
        N1 is  N-1,
        fibonacci( N1, F2, F1),
        F is  F1+ F2,!.
```

Uso:

SWI

```
?- fibonacci(6,F).    % fib(+,-)
F = 5.
?- fibonacci(6,5).    % fib(+,+)
true.
```

✎ **Comentário**
Nessa abordagem, para o cálculo de `fibonacci(40,F)`, o Prolog fará apenas 39 chamadas do predicado `fibonacci/2`.

* **Ex. 51 — (Torre de Hanói)** O matemático francês Edouard Lucas foi o criador do jogo Torre de Hanói, cujo nome é inspirado na torre símbolo da cidade de Hanói, no Vietnã. Há várias lendas a respeito da origem desse

jogo, a mais conhecida diz respeito a um templo Hindu, situado no centro do universo. Diz-se que Brahma supostamente havia criado uma torre com 64 discos de ouro e mais duas estacas equilibradas sobre uma plataforma. Brahma ordenara-lhes que movessem todos os discos da posição original para uma das estacas, segundo as suas instruções. As regras eram simples: apenas um disco poderia ser movido por vez e nunca um disco maior deveria ficar por cima de um disco menor. Segundo a lenda, quando todos os discos fossem todos transferidos para a estaca destino, o templo desmoronar-se-ia e o mundo desapareceria. Qual a solução para o problema da Torre de Hanói de 3 pinos para N discos?

Solução:

```
hanoi( N) :- hanoi( N, pino_1, pino_aux, pino_2).

hanoi(0, _ , _ , _ )     :- !.
hanoi( N, Origem, Aux, Destino) :-
    N1 is N-1,
    hanoi( N1, Origem, Destino, Aux),
    move( Origem, Destino),
    hanoi( N1, Aux, Origem, Destino).

move( I, F) :-    write('mover do '),write( I),
                    write(' para '),write( F), nl.
```

Uso:

SWI

```
?- hanoi(3).
mover do pino_1 para pino_2
mover do pino_1 para pino_aux
mover do pino_2 para pino_aux
mover do pino_1 para pino_2
mover do pino_aux para pino_1
mover do pino_aux para pino_2
mover do pino_1 para pino_2
true.
```

Ex. 52 — (Deleção de uma ocorrência de um elemento em uma lista) Como deletar uma ocorrência de um elemento em uma lista dada?

Solução:

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

```
deleta( X,[ H| T],[ H| R]) :- deleta( X, T, R).
```

Uso:

SWI

```
?- deleta(1,[2,1,2,1,3,4,1],X).    % deleta(+,+,-)
X = [2, 2, 1, 3, 4, 1] ;
X = [2, 1, 2, 3, 4, 1] ;
X = [2, 1, 2, 1, 3, 4] ;
false.
?- deleta(1,[2,1,2,1,3,4,1],[2,2,1,3,4,1]).    % deleta(+,+,+)
true
.
?- deleta(X,[2,1,2,1,3,4,1],[2,2,1,3,4,1]).    % deleta(-,+,+)
X = 1 ;
false.
```

Ex. 53 — (Deleção de todas as ocorrências de um elemento em uma lista) Como deletar todas as ocorrências de um elemento em uma lista dada?

Solução:

```
deleta_todos( X,[ X| T], R) :- !,  deleta_todos( X, T, R).
deleta_todos( X,[ H| T],[ H| R]) :- !,  deleta_todos( X, T, R).
deleta_todos(_,[ ],[ ]).
```

Uso:

SWI

```
?- deleta_todos(1,[2,1,2,1,3,4,1],X).    % deleta_todos(+,+,-)
X = [2, 2, 3, 4].
?- deleta_todos(1,[2,1,2,1,3,4,1],[2,2,3,4]).    % deleta_todos(+,+,+)
true.
```

Ex. 54 — (Substituição Parcial) Como substituir uma ocorrência de um elemento por outro elemento em uma lista dada?

Solução:

```
substitui( X, Y,[ X| T],[ Y| T]).
substitui( X, Y,[ H| T],[ H| R]) :- substitui( X, Y, T, R).
substitui(_,_,[ ],[ ]).
```

Uso:

SWI

```
?- substitui(1,a,[1,2,1,2,1,1],R).    % substitui(+,+,+,-)
R = [a, 2, 1, 2, 1, 1] ;
```

```
R = [1, 2, a, 2, 1, 1] ;
R = [1, 2, 1, 2, a, 1] ;
R = [1, 2, 1, 2, 1, a] ;
R = [1, 2, 1, 2, 1, 1] ;
false.
?- substitui(X,a,[1,2,1,2,1,1],[a,2,1,2,1,1]).    % substitui(-,+,+,+)
X = 1 ;
false.
?- substitui(1,Y,[1,2,1,2,1,1],[a,2,1,2,1,1]).    % substitui(+,-,+,+)
Y = a ;
false.
?- substitui(X,Y,[1,2,1,2,1,1],[a,2,1,2,1,1]).    % substitui(-,-,+,+)
X = 1,
Y = a ;
false.
?- substitui(1,a,[1,2,1,2,1,1],[a,2,1,2,1,1]).    % substitui(+,+,+,+)
true
.
```

Ex. 55 — (Substituição Total) Como substituir todas as ocorrências de um elemento por outro elemento em uma lista dada?

Solução:

```
substitui_todos( X, Y,[ X| T],[ Y| R]) :- !,  substitui_todos( X, Y, T, R).
substitui_todos( X, Y,[ H| T],[ H| R]) :- !,  substitui_todos( X, Y, T, R).
substitui_todos(_,_,[ ],[ ]).
```

Uso:

SWI

```
?- substitui_todos(1,a,[1,2,1,2,1,1],R).    % substitui_todos(+,+,+,-)
R = [a, 2, a, 2, a, a].
?- substitui_todos(X,a,[1,2,1,2],[a,2,a,2]).    % substitui_todos(-,+,+,+)
X = 1.
?- substitui_todos(1,Y,[1,2,1,2],[a,2,a,2]).    % substitui_todos(+,-,+,+)
Y = a.
?- substitui_todos(X,Y,[1,2,1,2],[a,2,a,2]).    % substitui_todos(-,-,+,+)
X = 1,
Y = a.
```

Ex. 56 — (Deleção do n-ésimo elemento de uma lista) a

Solução:

```
deleta_nesimo( N,[ H| T],[ H| R]) :-
    N > 1,
    N1 is  N-1,
    deleta_nesimo( N1, T, R).
deleta_nesimo(1,[_| T], T).
```

Uso:

SWI

```
?- deleta_nesimo(1,[a,b,c],X).    % deleta_nesimo(+,+,-)
X = [b, c].
?- deleta_nesimo(2,[a,b,c],X).    % deleta_nesimo(+,+,-)
X = [a, c].
?- deleta_nesimo(3,[a,b,c],[a,b]).    % deleta_nesimo(+,+,+)
true.
?- deleta_nesimo(3,X,[a,b,d]).    % deleta_nesimo(+,-,+)
X = [a, b, _G468, d].
```

Ex. 57 — (aaaa) Como selecionar o n-ésimo elemento de uma lista?

Solução:

```
pertence( H,[ H| _ ]).
```

Uso:

SWI

```
a
```

Ex. 58 — (Somatório (Versão 1)) Como calcular o somatório dos elementos de uma lista dada?

Solução:

```
somatorio([ H| T], S) :-
    somatorio( T, SomaT),
    S is  H +  SomaT.
somatorio([ ],0).
```

Uso:

SWI

```
?- somatorio([1,2,3,4],X).    % somatorio(+,-)
X = 10.
?- somatorio([1,2,3,4],10).    % somatorio(+,+)
true.
```

Ex. 59 — (Somatório Eficiente (Versão 2)) Como calcular o somatório dos elementos de uma lista dada de forma eficiente?

Solução:

```
somatorio2( Lista, S) :- somatorio2( Lista,0, S).

somatorio2([ ], Ac, Ac).
somatorio2([ H| T], Ac, S) :-
    NAc is Ac + H,
    somatorio2( T, NAc, S).
```

Uso:

SWI

```
?- somatorio2([1,2,3,4],X).   % somatorio2(+,-)
X = 10.
?- somatorio2([1,2,3,4],10).   % somatorio2(+,+)
true.
```

✎ **Comentário**
Neste exercício, uma demonstração do uso eficiente (recursão de cauda) de um acumulador.

Ex. 60 — (Produtório (Versão 1)) Como calcular o produtório dos elementos de uma lista dada?

Solução:

```
produtorio([ H| T], P) :-
    produtorio( T, ProdutoT),
    P is H * ProdutoT.
produtorio([ ],1).
```

Uso:

SWI

```
?- produtorio([1,2,3,4],X).   % produtorio(+,-)
X = 24.
?- produtorio([1,2,3,4],24).   % produtorio(+,+)
true.
```

Ex. 61 — (Produtório Eficiente (Versão 2)) Como calcular o produtório dos elementos de uma lista dada de forma eficiente?

Solução:

```
produtorio2( Lista, S) :- produtorio2( Lista,1, S ).

produtorio2([ ], Ac, Ac).
produtorio2([ H| T], Ac, S) :-
    NAc is Ac * H,
    produtorio2( T, NAc, S).
```

Uso:

SWI

```
?- produtorio2([1,2,3,4],X).    % produtorio2(+,-)
X = 24.
?- produtorio2([1,2,3,4],24).    % produtorio2(+,+)
true.
```

✎ **Comentário**
Neste exercício, uma demonstração do uso eficiente (recursão de cauda) de um acumulador.

Ex. 62 — (Somatório dos pares de uma lista) Qual é o somatório de todos os elementos pares de uma lista dada?

Solução:

```
soma_pares( Lista, Soma) :- soma_pares( Lista,0, Soma).

soma_pares([ ], Ac, Ac).
soma_pares([ H| T], Ac, Soma) :-
    0 is H mod 2,!,
    NAc is Ac+ H,
    soma_pares( T, NAc, Soma).
soma_pares([_| T], Ac, Soma) :-
    soma_pares( T, Ac, Soma).
```

Uso:

SWI

```
?- soma_pares([1,2,3,4,5,6,7,8],X).    %soma_pares(+,-)
X = 20.
?- soma_pares([1,2,3,4,5,6,7,8],20).   %soma_pares(+,+)
true.
```

Ex. 63 — (Somatório dos ímpares de uma lista) Qual é o somatório de todos os elementos ímpares de uma lista dada?

Solução:

```
soma_impares( Lista, Soma) :- soma_impares( Lista,0, Soma).

soma_impares([ ], Ac, Ac).
soma_impares([ H| T], Ac, Soma) :-
    0 is  H mod 2,!,
     NAc is  Ac+ H,
     soma_impares( T, NAc, Soma).
soma_impares([_| T], Ac, Soma) :-
     soma_impares( T, Ac, Soma).
```

Uso:

SWI

```
?- soma_impares([1,2,3,4,5,6,7,8],X).   %soma_impares(+,-)
X = 16.
?- soma_impares([1,2,3,4,5,6,7,8],20).   %soma_impares(+,+)
false.
```

Ex. 64 — (Intercalar elementos (zip)) Como intercalar os elementos de duas listas de mesmo tamanho?

Solução:

```
zip([  ],[  ],[  ]).
zip([ X| Xs],[ Y| Ys],[( X, Y)| R]) :-
     zip( Xs, Ys, R ),!.
```

Uso:

SWI

```
?- zip([1,2,3],[a,b,c],X).   % zip(+,+,-)
X = [ (1, a), (2, b), (3, c)].
?- zip(X,[a,b],[(1,a),(2,b)]).   % zip(-,+,+)
X = [1, 2].
?- zip([1,2],X,[(1,a),(2,b)]).   % zip(+,-,+)
X = [a, b].
?- zip([1,2],[a,b],[(1,a),(2,b)]).   % zip(+,+,+)
true.
?- zip(X,Y,[(1,a),(2,b)]).   % zip(-,-,+) como unzip
X = [1, 2],
Y = [a, b].
```

Comentário

A operação de `unzip/3` é realizada com o próprio predicado `zip/3`.

Ex. 65 — (Produto Interno) Sejam $\vec{v} = (v_x, v_y, v_z)$ e $\vec{w} = (w_x, w_y, w_z)$ dois vetores no $\mathbb{R}^3$. Qual é o produto interno entre $\vec{v}$ e $\vec{w}$?

Solução:

```
produto_interno( V, W, ProdInt) :-
    produto_interno( V, W,0, ProdInt).

produto_interno([ ],[ ], Ac, Ac).      % clásula de parada
produto_interno([ V| Vs],[ W| Ws], Ac, ProdInt) :-
    NAc is Ac + V* W,
    produto_interno( Vs, Ws, NAc, ProdInt).
```

Uso:

SWI

```
?- produto_interno([1,2,-1],[-3,0,-5],X).   % produto_interno(+,+,-)
X = 2.
?- produto_interno([1,2,-1],[-3,0,-5],2).   % produto_interno(+,+,+)
true.
```

Comentário
O predicado `produto_interno/3` pode receber vetores de qualquer dimensão, desde que $\vec{v}$ e $\vec{w}$ sejam de uma mesma dimensão.

Ex. 66 — (Remoção de elementos iniciais de uma lista) Dada uma lista qualquer, como retirar N elementos do seu início?

Solução:

```
trim(0, Lista, Lista) :- !.
trim( N,[_| T], Resp) :-
    NovoN is N-1,
    trim( NovoN, T, Resp).
```

Uso:

SWI

```
?- trim(3,[a,b,c,d,e,f],X).   % trim(+,+,-)
X = [d, e, f].
?- trim(3,[a,b,c,d,e,f],[d,e,f]).   % trim(+,+,+)
true.
```

Ex. 67 — (Remoção de elementos iniciais de uma lista (versão 2)) Dada uma lista qualquer, como retirar N elementos do seu início?

Solução:

```
trim2( N, Lista, Resp) :-
    concatena( Inicio, Resp, Lista ),
    comprimento( Inicio, N),!.
```

Uso:

SWI

```
?- trim2(3,[a,b,c,d,e,f],X).    % trim(+,+,-)
X = [d, e, f].
?- trim2(3,[a,b,c,d,e,f],[d,e,f]).    % trim(+,+,+)
true.
```

Ex. 68 — (Remoção de elementos finais de uma lista) Dada uma lista qualquer, como retirar N elementos do seu final?

Solução:

```
trim_ultimos( N, Lista, Resp) :-
    comprimento( Lista, Comp),
    Nro = Comp - N,
    seleciona_n_primeiros( Nro, Lista, Resp),!.
```

Uso:

SWI

```
?- trim_ultimos(3,[a,b,c,d,e,f],X).    % trim_ultimos(+,+,-)
X = [a, b, c].
?- trim_ultimos(3,[a,b,c,d,e,f],[a,b,c]).    % trim_ultimos(+,+,+)
true.
```

Comentário
Exemplo de aplicação do predicado `elimina_ultimo/2` (vide exercício 30).

Ex. 69 — (Remoção de elementos finais de uma lista (versçao 2)) Dada uma lista qualquer, como retirar N elementos do seu final?

Solução:

```
trim_ultimos2( N, Lista, Resp) :-
    concatena( Resp, Fim, Lista),
    comprimento( Fim, N),!.
```

Uso:

SWI
```
?- trim_ultimos2(3,[a,b,c,d,e,f],X).    % trim_ultimos(+,+,-)
X = [a, b, c].
?- trim_ultimos2(3,[a,b,c,d,e,f],[a,b,c]).    % trim_ultimos(+,+,+)
true.
```

Ex. 70 — (Máximo) Qual é o maior entre dois números?

Solução:

```
maximo( A, B, M) :- A > B,!, M= A.
maximo(_, B, B).
```

Uso:

SWI
```
?- maximo(3,2,X).    % maximo(+,+,-)
X = 3.
?- maximo(2,3,X).    % maximo(+,+,-)
X = 3.
?- maximo(2,3,2).    % maximo(+,+,+)
false.
?- maximo(2,3,3).    % maximo(+,+,+)
true.
```

✎ **Comentário**
Na primeira cláusula, `maximo(A,B,M) :- A > B,!,M=A.`, é fundamental fazer a unificação de M somente após a comparação entre os valores de `A` e `B`.

Ex. 71 — (Mínimo) Qual é o menor entre dois números?

Solução:

```
minimo( A, B, M) :- A < B,!, M= A.
minimo(_, B, B).
```

Uso:

SWI
```
?- minimo(3,2,X).    % minimo(+,+,-)
X = 2.
?- minimo(2,3,X).    % minimo(+,+,-)
X = 2.
?- minimo(2,3,2).    % minimo(+,+,+)
true.
```

```
?- minimo(2,3,3).    % minimo(+,+,+)
false.
```

Comentário
Exercício 70, *Mutatis Mutandis*.

Ex. 72 — (Máximo elemento de uma lista) Qual é o maior elemento de uma lista dada?

Solução:

```
maximo_lista([ H| T], Max) :-
    maximo_lista( T, H, Max).

maximo_lista([ ], Maior, Maior).
maximo_lista([ H| T], Maior, Max) :-
    H > Maior,!,
    maximo_lista( T, H, Max).
maximo_lista([_| T], Maior, Max) :-
    maximo_lista( T, Maior, Max).
```

Uso:

SWI

```
?- maximo_lista([10,3,5,20,19,17],M).    % maximo_lista(+,-)
M = 20.
?- maximo_lista([ ],M).    % maximo_lista(+,-)
false.
?- maximo_lista([10,3,5,20,19,17],20).    % maximo_lista(+,+)
true.
```

Comentário
O predicado `maximo_lista/2` para uma lista de N elementos faz exatamente $N-1$ comparações.

Ex. 73 — (Mínimo elemento de uma lista) Qual é o menor elemento de uma lista dada?

Solução:

```
minimo_lista([ H| T], Min) :-
    minimo_lista( T, H, Min).

minimo_lista([ ], Menor, Menor).
```

```
minimo_lista([ H| T], Menor, Min) :-
    H < Menor,!,
    minimo_lista( T, H, Min).
minimo_lista([_| T], Menor, Min) :-
    minimo_lista( T, Menor, Min).
```

Uso:

SWI

```
?- minimo_lista([10,3,5,20,19,17],M).   % minimo_lista(+,-)
M = 3.
?- maximo_lista([ ],M).   % minimo_lista(+,-)
false.
?- minimo_lista([10,3,5,20,19,17],3).   % minimo_lista(+,+)
true.
```

Ex. 74 — (Elementos máximo e mínimo de uma lista) Quais são os elementos máximo e mínimo de uma lista dada?

Solução:

```
max_min([ X], X, X).
max_min( Lista, Max, Min) :-
    divide( Lista , Metade1, Metade2),
    max_min( Metade1, Max1, Min1),
    max_min( Metade2, Max2, Min2),
    maximo( Max1, Max2, Max),
    minimo( Min1, Min2, Min).
```

Uso:

SWI

```
?- max_min([3,5,1,0,6,9,8,7,6,4],Max,Min).   % max_min(+,-,-)
Max = 9,
Min = 0.
?- max_min([3,5,1,0,6,9,8,7,6,4],9,Min).   % max_min(+,+,-)
Min = 0.
?- max_min([3,5,1,0,6,9,8,7,6,4],Max,0).   % max_min(+,-,+)
Max = 9.
?- max_min([3,5,1,0,6,9,8,7,6,4],9,0).   % max_min(+,+,+)
true.
```

2.3 Algoritmos baseados em relações de recorrência

** **Ex. 75 — (Cálculo de $\sqrt{x}$)**

Sejam $a_0, a_1, \ldots$ e $c_0, c_1, \ldots$ duas seqüências de números reais, definidas pelas relações de recorrência:

$$\begin{array}{l} a_i = a_{i-1} * (1 + \frac{1}{2} c_{i-1}) \\ c_i = c_{i-1}^2 * \frac{(3+c_{i-1})}{4} \end{array} \quad \text{para} i > 0$$

cujos valores iniciais são: $a_0 = 1$, $c_0 = 1 - x$ e $0 < x < 2$.
Por manipulação algébrica pode-se mostrar que $a_n = \sqrt{x * (1 - c_n)}$ e, uma vez que $|c_0| < 0$, segue-se que $\lim_{n \to \infty} c_n = 0$ e $\lim_{n \to \infty} a_n = \sqrt{x}$. Como determinar o valor de $\sqrt{x}$ com 8 casas decimais de precisão?

Solução:

```
/* raiz_quadrada(X,RaizX) :-
 RaizX é a raiz quadrada de X com 8 dígitos
  significativos , sendo 0<X<2. */

raiz_quadrada( X, RaizX) :-
      X > 0,  X < 2,
      C0 is 1- X,
      raiz_quadrada( X, C0, RaizX).

raiz_quadrada( A1, C1, RaizX) :- % cláusula de iteração
      A2 is  A1 * (1+ C1/2),
      C2 is  C1^2*(3+ C1)/4,
      Dif is  abs( A2- A1),
      Dif => 0.0000001,
      raiz_quadrada( A2, C2, RaizX).
raiz_quadrada( A1, C1, A2) :-   % cláusula de parada!
      A2 is  A1 * (1+ C1/2),
      C2 is  C1^2*(3+ C1)/4,
      Dif is  abs( A2- A1),
      Dif < 0.0000001.
```

Uso:

SWI

```
?- raiz_quadrada(1.2,RaizX)    % raiz_quadrada(+,-)
X =
?- raiz_quadrada(1.2,RaizX)    % raiz_quadrada(+,+)
X =
```

2.4 Lógica

Ex. 76 — (If...Then) Como implementar uma estrutura de programação tipo If...Then para ser usada na implementação de outros predicados Prolog?

Solução:

```
ifthen( P, Q) :- P,!, Q.
ifthen(_,_).
```

Uso:

SWI

```
?- ifthen(1 > 2, write(a)).    % ifthen(+,+)
true.
?- ifthen(1 < 2, write(a)).    % ifthen(+,+)
a
true.
```

Ex. 77 — (If...Then...Else) Como implementar uma estrutura de programação tipo If...Then...Else para ser usada na implementação de outros predicados Prolog?

Solução:

```
ifthenelse( P, Q,_) :- P,!, Q.
ifthenelse(_,_, R) :- R.
```

Uso:

SWI

```
?- ifthenelse(1 > 2, write(a), write(b)).    % ifthenelse(+,+,+)
b
true.
?- ifthenelse(1 < 2, write(a), write(b)).    % ifthenelse(+,+,+)
a
true.
```

Ex. 78 — (Literal) Seja L uma lógica proposicional clássica com os conectivos $\{\neg, \rightarrow, \leftrightarrow, \vee, \wedge\}$. Dada uma string $\mathcal{S}$ como determinar se $\mathcal{S}$ é um literal?

Solução:

```
literal ( F)       :- atomic( F).
literal ( nao  F) :- atomic( F).
```

Uso:

SWI

```
?- literal(p).    % literal(+)
true.
?- literal(nao p).    % literal(+)
true.
?- literal(q => p).    % literal(+)
false.
?- literal(nao(q => p)).    % literal(+)
false.
```

Comentário
O conceito de literal é relativo ao alfabeto da lógica considerada. Dado o conjunto das variáveis proposicionais V, se $v \in V$ então v e $\neg v$ são literais.

2.4.1 Formas Normais

*** **Ex. 79 — (Forma Normal Conjuntiva)** Seja L uma lógica proposicional clássica com os conectivos $\{\neg, \rightarrow, \leftrightarrow, \vee, \wedge\}$. Dada uma fórmula bem-formada $\mathcal{F}$ como determinar uma forma normal conjuntiva de $\mathcal{F}$?

Solução:

```
:- op(100, fy, nao).
:- op(110, xfy,^).
:- op(115, xfy, v).
:- op(120, xfy,=>).
:- op(130, xfy,<=>).

fnc( F ^  G, Res1 ^  Res2) :-
     !,
     fnc( F, Res1),
     fnc( G, Res2).
fnc( Formula, Resultado) :-
     transforma( Formula, FormulaNova),
     !,
     ifthenelse ( Formula =  FormulaNova,
                          Resultado =  FormulaNova,
```

```
                    fnc( FormulaNova, Resultado)).

transforma( nao( nao  X), X).
transforma( nao( X =>  Y),  X ^  nao  Y).
transforma( X <=>  Y, ( nao  X  v  Y) ^ ( nao  Y  v  X)).
transforma( X =>  Y,  nao  X  v  Y).
transforma( nao( X ^  Y), nao  X  v  nao  Y).
transforma( nao( X  v  Y), nao  X ^  nao  Y).
transforma( X ^ ( Y  v  Z),( X ^  Y)  v ( X ^  Z)).
transforma( X  v ( Y ^  Z),( X  v  Y) ^ ( X  v  Z)).
transforma(( X ^  Y)  v  Z,( X  v  Z) ^ ( Y  v  Z)).
transforma(( X  v  Y) ^  Z,( X ^  Z)  v ( Y ^  Z)).
transforma( X  v  Y, ( X1  v  Y1)) :-
            transforma( X, X1),
            transforma( Y, Y1).
transforma( nao  X, nao  X1) :-
            transforma( X, X1).
transforma( X, X) :-  literal ( X).
```

Uso:

SWI

```
?- fnc(a => (b ^ a),FNC).    % fnc(+,-)
FNC = (nao a v b)^ (nao a v a).
?- fnc(nao (a ^ b) => (b ^ a),FNC).    % fnc(+,-)
FNC = ((a v b)^ (b v b))^ (a v a)^ (b v a).
```

Comentários

- Os conectivos $\{\neg, \rightarrow, \leftrightarrow, \vee, \wedge\}$ são representados no programa respectivamente por os conectivos {nao, =>, <=>, v, ^};
- Os predicados `ifthenelse/3` e `literal` estão implementados nos exercícios 77 e 78 respectivamente.

*** **Ex. 80 — (Forma Normal Disjuntiva)** Seja L uma lógica proposicional clássica com os conectivos $\{\neg, \rightarrow, \leftrightarrow, \vee, \wedge\}$. Dada uma fórmula bem-formada $\mathcal{F}$ como determinar uma forma normal disjuntiva de $\mathcal{F}$?

Solução:

```
:- op(100, fy, nao).
:- op(110, xfy, v).
:- op(115, xfy,^).
```

```
:- op(120, xfy,=>).
:- op(130, xfy,<=>).

fnd( F v G, Res1 v Res2) :-
    !,
    fnd( F, Res1),
    fnd( G, Res2).
fnd( Formula, Resultado) :-
    transforma2( Formula, FormulaNova),
    !,
    ifthenelse( Formula = FormulaNova,
                Resultado = FormulaNova,
                fnd( FormulaNova, Resultado)).

transforma2( nao( nao X), X).
transforma2( nao( X => Y), X ^ nao Y).
transforma2( X <=> Y, ( nao X v Y) ^ ( nao Y v X)).
transforma2( X => Y, nao X v Y).
transforma2( nao( X ^ Y), nao X v nao Y).
transforma2( nao( X v Y), nao X ^ nao Y).
transforma2( X ^ ( Y v Z),( X ^ Y) v ( X ^ Z)).
transforma2( X v ( Y ^ Z),( X v Y) ^ ( X v Z)).
transforma2(( X ^ Y) v Z,( X v Z) ^ ( Y v Z)).
transforma2(( X v Y) ^ Z,( X ^ Z) v ( Y ^ Z)).
transforma2( X ^ Y, ( X1 ^ Y1)) :-
        transforma2( X, X1),
        transforma2( Y, Y1).
transforma2( nao X, nao X1) :-
        transforma2( X, X1).
transforma2( X, X) :- literal ( X).
```

Uso:

SWI

```
?- fnd(a => (b ^ a),FNC).    % fnd(+,-)
FNC = nao a v (b^a).
?- fnd(nao(a ^ b) => (b ^ a),FNC).    % fnd(+,-)
FNC = (a^b)v (b^a).
```

✎ Comentários

•Os conectivos $\{\neg, \rightarrow, \leftrightarrow, \vee, \wedge\}$ são representados no programa respecti-

vamente por os conectivos {nao, =>, <=>, v, ^};

- Os predicados `ifthenelse/3` e `literal` estão implementados nos exercícios 77 e 78 respectivamente.

2.4.2 Unificação

Ex. 81 — (Termo composto) Como determinar se um termo da Lógica de Predicados de 1^a Ordem é ou não composto?

Solução:

```
termo_composto( X) :-
    functor( X,_, N),
     N>0.
```

Uso:

SWI

```
?- termo_composto(f(1,2,a,g(3))).   % termo_composto(+).
true.
?- termo_composto(f(1)).   % termo_composto(+).
true.
?- termo_composto(p(1,2)).   % termo_composto(+).
true.
?- termo_composto(X).   % termo_composto(+).
false.
?- termo_composto(a).   % termo_composto(+).
false.
```

* **Ex. 82 — (Não ocorrência de variável em termo)** No domínio da Lógica de Predicados de 1^a Ordem, como determinar se uma variável não ocorre em um termo?

Solução:

```
nao_ocorre( X, Y) :- var( Y),  X \==  Y.
nao_ocorre( X, Y) :- nonvar( Y), atomic( Y).
nao_ocorre( X, Y) :-
        nonvar( Y),
         termo_composto( Y),
        functor( Y, F, N),
         nao_ocorre( N, X, Y).
```

```
nao_ocorre( N, X, Y) :-
          N>0,
         arg( N, Y, Arg),
          nao_ocorre( X, Arg),
          N1is  N-1,
          nao_ocorre( N1, X, Y),!.
nao_ocorre(0,_,_) :-  !.
```

Uso:

SWI

```
?- nao_ocorre(X,f(a,1)).    % não_ocorre(-,+)
true.
?- nao_ocorre(X,f(a,1,g(X))).    % não_ocorre(-,+)
false.
?- nao_ocorre(X,f(a,1,g(2,Y))).    % não_ocorre(-,+)
true.
?- nao_ocorre(X,f(a,1,g(2,X))).    % não_ocorre(-,+)
false.
```

Comentários

- O predicado `nao_ocorre/2` tem sempre como primeiro argumento uma variável, em Prolog, obrigatoriamente iniciada com letra maiúscula.
- O predicado `termo_composto/1` está implementado no exercício 81.
- `Termo1 \== Termo2` é verificado quando Termo1 não é idêntico ao Termo2, é equivalente a `(\+Termo1 == Termo2)`.

*** **Ex. 83 — (Unificação, com checagem de ocorrência)** Como unificar dois termos da Lógica de Predicados de 1^a Ordem?

Solução:

```
/*
  unifica(Termo1,Termo2) :-
     Termo1 e Termo2 sao unificados com checagem de ocorrência.
*/
unifica( X, Y) :- var( X), var( Y),  X= Y, !.
unifica( X, Y) :- var( X), nonvar( Y), nao_ocorre( X, Y),  X= Y, !.
unifica( X, Y) :- nonvar( X), var( Y),  nao_ocorre( Y, X),  Y= X, !.
unifica( X, X) :- nonvar( X), atomic( X), !.
unifica( X, Y) :-
```

```
        nonvar( X),
        nonvar( Y),
         termo_composto( X),
         termo_composto( Y),
         unifica_termo( X, Y),!.

unifica_termo( X, Y) :-
        functor( Y, F, N),
        functor( X, F, N),
         unifica_argumentos( N, X, Y).

unifica_argumentos( N, X, Y) :-
         N>0,
         unifica_arg( N, X, Y),
         N1 is  N-1,
         unifica_argumentos( N1, X, Y).
unifica_argumentos(0,_,_).

unifica_arg( N, X, Y) :-
        arg( N, X, ArgX),
        arg( N, Y, ArgY),
         unifica( ArgX, ArgY).
```

Uso:

SWI

```
?- unifica(X,f(1)).
X = f(1).
?- unifica(f(X),f(1)).
X = 1.
?- unifica(f(X,Y),f(1,X)).
X = 1,
Y = 1.
?- unifica(f(a,Y),f(1,X)).
false.
?- unifica(f(1,Z,Y),f(X,1,a)).
Z = 1,
Y = a,
X = 1.
?- unifica(f(1,g(Z),a),f(X,g(f(1)),a)).
Z = f(1),
X = 1.
?- unifica(f(1,g(Z),b),f(X,g(f(1)),a)).
```

```
false.
?- unifica(X,f(X)).
false.
```

✎ **Comentário**
Os predicados `termo_composto/1` e `nao_ocorre/2` estão implementados nos exercícios 81 e 82 respectivamente.

*** Ex. 84 — **(Unificação, sem checagem de ocorrência)** Como unificar dois termos da Lógica de Predicados de 1^a Ordem sem efetuar a checagem de ocorrência de uma variável em um termo?

Solução:

```
unifica_sco( X, Y) :- var( X), var( Y),  X= Y, !.
unifica_sco( X, Y) :- var( X), nonvar( Y),  X= Y, !.
unifica_sco( X, Y) :- nonvar( X), var( Y),  Y= X, !.
unifica_sco( X, X) :-
        nonvar( X), nonvar( Y),
        atomic( X), atomic( Y),
         X= Y.
unifica_sco( X, Y) :-
        nonvar( X),
        nonvar( Y),
         termo_composto( X),
         termo_composto( Y),
         unifica_termo_sco( X, Y), !.

unifica_termo( X, Y) :-
        functor( Y, F, N),
        functor( X, F, N),
         unifica_argumentos( N, X, Y).

unifica_argumentos( N, X, Y) :-
         N>0,
         unifica_arg( N, X, Y),
         N1 is  N-1,
         unifica_argumentos( N1, X, Y).
unifica_argumentos(0,_,_).

unifica_arg( N, X, Y) :-
```

```
        arg( N, X, ArgX),
        arg( N, Y, ArgY),
         unifica( ArgX, ArgY).
```

Uso:

SWI

```
?- unifica(X,f(1)).
X = f(1).
?- unifica(f(X),f(1)).
X = 1.
?- unifica(f(X,Y),f(1,X)).
X = 1,
Y = 1.
?- unifica(f(a,Y),f(1,X)).
false.
?- unifica(f(1,Z,Y),f(X,1,a)).
Z = 1,
Y = a,
X = 1.
?- unifica(f(1,g(Z),a),f(X,g(f(1)),a)).
Z = f(1),
X = 1.
?- unifica(f(1,g(Z),b),f(X,g(f(1)),a)).
false.
?- unifica(X,f(X)).
X = f(**).          % aqui está o problema em não checar ocorrências
```

Comentário

- Os compiladores e interpretadores Prolog implementam o algoritmo de unificação sem checagem de ocorrência para melhorar o desempenho dos programas. Todavia, a tarefa de verificar a ocorrência de variáveis em termos. Faça o teste no SWI-Prolog.
- Complementarmente, no SWI-Prolog está implementada a unificação correta (`unify_with_occurs_check/2`), com checagem de ocorrência. Uma variável somente é unificada com um termo se nesse termo não há ocorrência da própria variável.

SWI

```
?- unify_with_occurs_check(X, f(X)).
false.
```

2.5 Estatística e Probabilidade

Ex. 85 — (Média (Versão 1)) Qual é a média dos elementos de uma lista?

Solução:

```
media([ H| T], M) :-
    comprimento([ H| T], C),
    somatorio([ H| T], S),
    M is  S/ C.
```

Uso:

SWI

```
?- media([1,2,3,4],X).    % media(+,-)
X = 2.5.
?- media([1,2,3,4],2.5).    % media(+,+)
true.
```

* **Ex. 86 — (Média Eficiente (Versão 2))** Qual é a média dos elementos de uma lista?

Solução:

```
media2([ H| T], M) :-  media2([ H| T],0,0, M).

media2([ ], AcSoma, AcCont, M) :-  M is  AcSoma/ AcCont.
media2([ H| T], AcSoma, AcCont, M) :-
    NAcSoma is  AcSoma+ H,
    NAcCont is  AcCont + 1,
    media2( T, NAcSoma, NAcCont, M).
```

Uso:

SWI

```
?- media2([1,2,3,4],X).    % media2(+,-)
X = 2.5.
?- media2([1,2,3,4],2.5).    % media2(+,+)
true.
```

Comentário

Neste exercício, uma demonstração do uso eficiente (recursão de cauda) de dois acumuladores (uma para acumular o somatório dos elementos da lista, outro para contar quantos elementos existem na lista).

* **Ex. 87 — (Mediana)** Qual é a mediana de uma população representada por uma lista ordenada crescente?

Solução:

```
mediana([ M], M) :- !.
mediana([ A, B], M) :- !,  M is ( A+ B)/2.
mediana([_| T], M) :-
    elimina_ultimo( T, NovaLista),
    mediana( NovaLista, M), !.
```

Uso:

SWI

```
?- mediana([1,2,4,5,9],M).    % mediana(+,-)
M = 4.
?- mediana([1,2,4,5,9],4).    % mediana(+,+)
true.
?- mediana([1,4,5,9],M).    % mediana(+,-)
M = 4.5.
?- mediana([1,4,5,9],4.5).    % mediana(+,+)
true.
```

Comentário

O predicado `elimina_ultimo/2` está implementado no exercício 30.

** **Ex. 88 — (Frequência de uma população)** Qual é a frequencia dos elementos de uma população representada por uma lista de valores?

Solução:

```
frequencia_populacao([ P| Ps], Freq) :-
    frequencia_populacao( Ps,[[ P,1]], Freq).

frequencia_populacao([ H| T], Ac, Freq) :-
    insere_frequencia( H, Ac, NovoAc),
    frequencia_populacao( T, NovoAc, Freq),!.
frequencia_populacao([ ], Ac, Ac) :-  !.

insere_frequencia( Elem,[[ H, N]| T],[[ H, N]| R]) :-
    Elem @>  H,!,
    insere_frequencia( Elem, T, R).
insere_frequencia( Elem,[[ Elem, N]| T],[[ Elem, N1]| T]) :-  N1 is  N+1.
insere_frequencia( Elem, T,[[ Elem,1]| T]) :-  !.
```

Uso:

SWI

```
?- frequencia_populacao([1,2,1,2,1,2,3,3,4],X).  % frequencia_populacao(+,-)
X = [[1, 3], [2, 3], [3, 2], [4, 1]].
?- frequencia_populacao([a,b,c,d,a,c,d,e,a],X).  % frequencia_populacao(+,-)
X = [[a, 3], [b, 1], [c, 2], [d, 2], [e, 1]].
?- frequencia_populacao([1,2,2],[[1,1],[2,2]]).  % frequencia_populacao(+,+)
true.
```

** **Ex. 89 — (Ordenação de lista de frequências)** Como ordenar, em ordem decrescente, uma lista contendo a frequencia dos elementos de uma população representada por uma lista de valores?

Solução:

```
ordena_frequencia( Freq, FreqOrdenada) :-
    ordena_frequencia( Freq,[ ], FreqOrdenada),!.

ordena_frequencia([ NF| T], Ac, FreqOrd) :-
    insere_frequencia_ord( NF, Ac, NAc),
    ordena_frequencia( T, NAc, FreqOrd).
ordena_frequencia([ ], Ac, Ac) :-  !.

insere_frequencia_ord([ N, F],[[ N1, F1]| T],[[ N, F],[ N1, F1]| T]) :-
    F >= F1,!.
insere_frequencia_ord([ N, F],[[ N1, F1]| T],[[ N1, F]| R]) :-
    !,
    insere_frequencia_ord([ N, F], T, R).
insere_frequencia_ord(_,[ ],[ ]).
```

Uso:

SWI

```
?- ordena_frequencia([[1,1],[2,1],[3,4],[4,3]],X).  % ordena_frequencia(+,-)
X = [[3, 4], [4, 3], [2, 1], [1, 1]].
```

Comentário
Ordenação pelo método de inserção direta (exercício **??**).

*** **Ex. 90 — (Moda)** Qual é a moda de uma população representada por uma lista de valores?

Solução:

```
moda( Populacao, MODA) :-
    frequencia_populacao( Populacao, Freq),
    ordena_frequencia( Freq, FreqOrdenada),
    Freq = [[_, F]|_],
    F > 1,!,
    maiores_frequencias( FreqOrdenada, MODA).
moda(_,[ ]) :- !.

maiores_frequencias([[ N1, F1],[ N2, F2]|_],[[ N1, F1]]) :-
    F2 < F1,!.
maiores_frequencias([[ N1, F],[ N2, F]| T],[[ N1, F]| R]) :-
    maiores_frequencias([[ N2, F]| T], R),!.
maiores_frequencias([[ N, F]],[[ N, F]]) :- !.
```

Uso:

SWI

```
?- moda([1,2,3,4,5,6],X).    % moda(+,-)
X = [].
?- moda([1,2,1,2,1,2,3,4,5],X).    % moda(+,-)
X = [[2, 3], [1, 3]].
?- moda([a,b,c,a,b,a],X).    % moda(+,-)
X = [[a, 3]].
```

Comentário

Em estatística descritiva, o valor ou valores que detém o maior número de observações, ou seja, os de maior frequência são denominado *Moda*. Ao contrário da média ou mediana, a Moda não é necessariamente única. A moda da lista de valores $[1,1,2,2,2,3,4,4]$ é 2, $[1,1,3,2,2,3,4,4]$ apresenta quatro modas (1, 2, 3 e 4) e $[a,b,c,d,e,f]$ não possui moda.

** **Ex. 91** — **(Desvio Padrão Amostral)** Qual é o desvio padrão amostral de uma população representada por uma lista de valores?

Solução:

```
desvio_padrao( Populacao, DP) :-
    media( Populacao, Media),
    comprimento( Populacao, N),
    soma_quadrados_dif( Populacao, Media, SomaQuad),
    DP is sqrt( SomaQuad/( N-1)).
```

```
soma_quadrados_dif([ H| T], Media, SomaQuad) :-
    soma_quadrados_dif([ H| T], Media,0, SomaQuad),!.

soma_quadrados_dif([ ],_, Ac, Ac) :- !.
soma_quadrados_dif([ H| T], Media, Ac, SomaQuad) :-
    NAc is ( H- Media)*( H- Media)+ Ac,
    soma_quadrados_dif( T, Media, NAc, SomaQuad).
```

Uso:

SWI

```
?- desvio_padrao([1,2,3,4,5,6,7,7,8,8,8],C).   % desvio_padrao(+,-)
C = 2.54058.
```

2.6 Tempo e Datas

Ex. 92 — (Ano Bissexto) Como determinar se um ano é ou não bissexto?

Solução:

```
ano_bissexto( Ano) :-
    (( Ano mod 4 =:= 0,  Ano mod 100 =\= 0) ; % ou-lógico
       Ano mod 400 =:= 0), !.
```

Uso:

SWI

```
?- ano_bissexto(2004).   % ano_bissexto(+)
true.
?- ano_bissexto(1000).   % ano_bissexto(+)
false.
?- ano_bissexto(2000).   % ano_bissexto(+)
true.
```

Comentário

Neste exercício usa-se os predicados aritméticos "=:=´´ e "=\=´´. A =:= B é verdadeiro se a avaliação da expressão A é igual à avaliação da expressão B e A =\= B é verdadeiro se a avaliação da expressão A é distinta da avaliação da expressão B.

Ex. 93 — (Número de dias passados em um ano (Versão 1)) Quantos dias já se passaram nesse ano?

Solução:

```
dias_desde_inicio_do_ano( Dia/ Mes/ Ano, NroDias) :-
    dia_bissexto( Mes/ Ano, DiaBissexto), % há dia bissexto nessa data?
    dias_mes_anteriores( Mes, DiasAnteriores),
    NroDias is DiasAnteriores + Dia + DiaBissexto ,!.

dia_bissexto( Mes/ Ano,1) :- ano_bissexto( Ano), Mes > 2,!.
dia_bissexto(_,0) :- !.

dias_mes_anteriores( Mes, DiasAnteriores) :-
    pertence([ Mes, DiasAnteriores ],[[1,0],[2,31],[3,59],
                                      [4,90],[5,120],[6,151],
                                      [7,181],[8,212],[9,243],
                                      [10,273],[11,304],[12,334]]).
```

Uso:

SWI

```
?- dias_desde_inicio_do_ano(9/12/2010,X). % dias_desde_inicio_do_ano(+,-)
X = 343.
?- dias_desde_inicio_do_ano(21/12/2012,X). % dias_desde_inicio_do_ano(+,-)
X = 356.
?- dias_desde_inicio_do_ano(1/3/2012,X). % dias_desde_inicio_do_ano(+,-)
X = 61.
?- dias_desde_inicio_do_ano(1/3/2011,X). % dias_desde_inicio_do_ano(+,-)
X = 60.
?- dias_desde_inicio_do_ano(1/3/2011,60). % dias_desde_inicio_do_ano(+,+)
true.
```

Comentário
Este exercício foi implementado no dia 09/10/2010.

Ex. 94 — (Diferença entre duas datas) Qual a diferença em dias entre duas datas dadas?

Solução:

```
dif_datas( D1/ M1/ A1, D2/ M2/ A2, Dias) :-
    data_dias( D1/ M1/ A1, DataDias1), % quantos dias se passaram desde
    data_dias( D2/ M2/ A2, DataDias2), % a data hipotética 01/03/00
    Dias is abs( DataDias1- DataDias2).

dias_data( DataDias, Dia/ Mes/ Ano) :-
```

```
    A is (10000* DataDias + 14780)//3652425,
    DAux is DataDias - 365* A + A//4 - A//100 + A//400,
    calculaDDD( DataDias, A, DAux, DDD, AnoA),
    Mi is (100* DDD + 52)//3060,
    Mes is ( Mi + 2) mod 12 + 1,
    Ano is AnoA + ( Mi + 2)//12,
    Dia is DDD - ( Mi * 306 + 5)//10 + 1.

calculaDDD( DataDias, A, DAux, DDD, AnoA) :-
    DAux < 0, !,
    AnoA is Ano-1,
    DDD is DataDias-(365* AnoA+ AnoA//4- AnoA//100+ AnoA//400).
calculaDDD( DataDias, A,_, DDD, A) :-
    DDD is DataDias-(365* A+ A//4- A//100+ A//400),!.
```

Uso:

SWI

```
?- dif_datas(1/1/2010,3/1/2010,X).   % dif_datas(+,+,-)
X = 2.
?- dif_datas(3/1/2010,1/1/2010,X).   % dif_datas(+,+,-)
X = 2.
?- dif_datas(1/2/2012,1/4/2012,X).   % dif_datas(+,+,-)
X = 60.
```

Comentário

Para compreensão do método de cálculo visite o sítio:
http://alcor.concordia.ca/~gpkatch/gdate-method.html.

Ex. 95 — (Número de dias passados em um ano (Versão 2)) Quantos dias já se passaram nesse ano?

Solução:

```
dias_desde_inicio_do_ano2( Dia/ Mes/ Ano, NroDias) :-
    dif_datas(01/01/ Ano, Dia/ Mes/ Ano, Dias),
    NroDias is Dias+1.
```

Uso:

SWI

```
?- dias_desde_inicio_do_ano2(9/12/2010,X). % dias_desde_inicio_do_ano(+,-)
X = 343.
?- dias_desde_inicio_do_ano2(21/12/2012,X). % dias_desde_inicio_do_ano(+,-)
X = 356.
```

```
?- dias_desde_inicio_do_ano2(1/3/2012,X). % dias_desde_inicio_do_ano(+,-)
X = 61.
?- dias_desde_inicio_do_ano2(1/3/2011,X). % dias_desde_inicio_do_ano(+,-)
X = 60.
?- dias_desde_inicio_do_ano2(1/3/2011,60). % dias_desde_inicio_do_ano(+,+)
true.
```

Comentários

- Este exercício foi implementado no dia 09/10/2010.
- Este exercício usa o predicado `dif_data/3` implementado no exercício 94.

Ex. 96 — ()

Solução:

Uso: SWI

```
a
```

Referências Bibliográficas

[1] L. Lamport. **LaTeX A Document Preparation System** Addison-Wesley, California 1986.