```
/*-------------------------------------------------------------*/
% Exame de PPIII - 1a. chamada de 06/07
%            PROPOSTA DE RESOLUCAO
%
/*-------------------------------------------------------------*/

% 1.

% a) NÚMERO DE DESSITENTES



desistentes([],0).
desistentes(L,N):- length(L,Tp), totalInscritos(Num), N is Num - Tp.


% b) PRIMEIRO POR TIPO

% COM UTILIZAÇÃO DE META-PREDICADOS

% dada a lista dos tempos de um piloto soma os valores dos tempos
tempoTotalPiloto(L,Tempo):- sumlist(L,Tempo).

primeiroPorTipo(L,Tipo,Nome/Numero/Pais):-
   setof(Tempo/Nome/Numero/Pais,
        LT^(member(piloto(Nome,Numero,Tipo,Pais,LT),L),tempoTotalPiloto(LT,Tempo)),
        [_/Nome/Numero/Pais|_]).

% COM UTILIZAÇÃO DE RECURSIVIDADE

primeiroPorTipo1(L,Tipo,Nome/Numero/Pais):-
   aux(L,[],Lr),
   sort(Lr,Lr2),
   aux2(Lr2,Tipo,Nome/Numero/Pais).


aux([],Ac,Ac).
aux([piloto(Nome,Numero,Tipo,Pais,Lt)|Resto],Lac,Lr):-
    tempoTotalPiloto(Lt,Tempo),
    append(Lac,[Tempo/Nome/Numero/Tipo/Pais],Lac1
    aux(Resto,Lac1,Lr).


aux2([Tempo/Nome/Numero,Tipo,Pais|Resto],Tipo,Nome/Numero/Pais):- !.
aux2([H|Resto],Tipo,Nome/Numero/Pais):- aux2(Resto,Tipo,Nome/Numero/Pais).




% c) ESCREVER NA BC

% COM UTILIZAÇÃO DE META-PREDICADOS

escreve(L):-
   findall(_, (member(piloto(Nome,Numero,Tipo,Pais,LT),
               tempoTotalPiloto(LT,Tempo),
               assert(tempo_final(Nome,Numero,Tipo,Tempo)),
          _).

% COM UTILIZAÇÃO DE RECURSIVIDADE
```

```
escreve1([]):- !.
escreve1([piloto(Nome,Numero,Tipo,Pais,LT))|Resto]) :-
   tempoTotalPiloto(LT, Tempo),
   assert(tempo_final(Nome,Numero,Tipo,Tempo)),
   escreve1(Resto).

% 2.

% aluno(num, nome, curso/ano/inscricao) sendo inscricao uma [cod]
% disciplina(cod, curso, ano)

aluno(1, a, lesi/1/[1,2,4]).
aluno(2, b, lesi/2/[2,3,4,9]).
aluno(3, x, lesi/1/[1,2,9]).
aluno(11, d, lesi/2/[2,3,9]).
aluno(111, j, lesi/3/[2,3,4,10,11]).
aluno(12, l, lesi/3/[3,10,11]).
aluno(10, s, lesi/3/[1,4,11]).
aluno(100, r, lesi/2/[7,8,9]).
aluno(110, oo, lesi/2/[7,8,9]).
aluno(13, all, lesi/2/[7,8,9]).
aluno(131, ag, lmcc/1/[5,6]).
aluno(145, hh, lmcc/2/[5,8,9]).
aluno(166, bb, lmcc/2/[6,8,9]).
aluno(17, ff, lmcc/2/[8,9]).
aluno(177, xx, lmcc/2/[5,9]).

disciplina(1, lesi, 1).
disciplina(2, lesi, 1).
disciplina(3, lesi, 2).
disciplina(4, lesi, 2).
disciplina(5, lmcc, 1).
disciplina(6, lmcc, 1).
disciplina(7, lmcc, 2).
disciplina(8, lmcc, 2).
disciplina(9, lesi, 2).
disciplina(10, lesi, 3).
disciplina(11, lesi, 3).
disciplina(12, lmcc, 3).


% a) Quantos alunos frequentam a disciplina D.

frequentam(D, N) :- findall(Num, (aluno(Num, _, _/_/Insc),
                                  member(D, Insc)), Lnal),
                            length(Lnal, N).

frequentam1(D, N) :- setof(Num, Nm^C^A^Insc^(aluno(Num, Nm, C/A/Insc),
                                 member(D, Insc)), Lnal),
                               length(Lnal, N).

% b) Qual a disciplina do ano A com mais alunos inscritos.

maisInscDiscAno(A, CodDiscp) :- setof(Cod, disciplina(Cod, _, A), Lda), !,
                                setof(Nal/D, (frequentam1(D, Nal),
                                              member(D, Lda)), LpNadiscp),
                                last(_/CodDiscp, LpNadiscp).
maisInscDiscAno(_, 0).
% ou simplesmente ...

maisInscDiscAno1(A, CodDiscp) :- setof(Num/Cod,
                                       (disciplina(Cod, _, A),
```

```
                                        frequentam1(Cod, Num)), Lda), !,
                                  last(_/CodDiscp, Lda).
maisInscDiscAno1(_, 0).


% c) Lista de pares Ano/Nal para um dado curso.

anosDeUmCurso(Cc, Lac) :- setof(Ano, N^Nm^Insc^aluno(N, Nm, Cc/Ano/Insc), Lac).
anosDeUmCurso(_, []).

numAlAnoCurso(Ano, Curso, Numal) :- setof(N, Nom^I^aluno(N, Nom, Curso/Ano/I), Ln),
                                    length(Ln, Numal), !.
numAlAnoCurso(_,_, 0).

paresAnoNumAlPorCurso(Cc, Lp) :- anosDeUmCurso(Cc, La),
                                 setof(An/Numal, (member(An, La),
                                          numAlAnoCurso(An, Cc, Numal)), Lp).

% ou de forma mais directa mas menos legivel ...

paresAnoNumAlPorCurso1(Cc, Lac, Lp) :-
              setof(Ano, N^Nm^Insc^aluno(N, Nm, Cc/Ano/Insc), Lac),
              setof(Ano/NumAlCurso,
                  ( member(Ano, Lac), (setof(N, I^Nom^aluno(N, Nom, Cc/Ano/I), Ln),
              length(Ln, NumAlCurso)) ), Lp).

% d) Eliminar todas as disciplinas com codigos na lista parametro.

:- dynamic(disciplina/3).

eliminaDiscp([]) :- !.
eliminaDiscp([Cd|Lcd]) :- retract(disciplina(Cd, _, _)), eliminaDiscp(Lcd).

% ou melhor ainda

eliminaDiscp1(Lcd) :- findall(_, (member(C, Lcd),
                                  retract(disciplina(C, _, _)), _).


% e) Ler de um ficheiro novas inscricoes de alunos e acrescentar ou alterar
% cf. o codigo do aluno já exista na BC ou seja novo.

:-dynamic(aluno/3).

codsAlunos(Lca) :- !, findall(Ca, aluno(Ca, _, _), Lca).
codsAlunos([]).

/* Ler os termos de um ficheiro F e carregar a BC usando REPEAT */

lerAlunos(F):- seen, see(F),
               codsAlunos(Lca),
               repeat,
                  read(T),
                  (T == end_of_file, seen, !
                   ;
                   tratar(T, Lca), fail).


tratar(aluno(C, N, I), Lc) :- member(C, Lc), !, retract(aluno(C, _, _)),
                              assert(aluno(C, N, I)), !.
tratar(aluno(C, N, I), _) :- assert(aluno(C, N, I)).
```

/* Versão recursiva; */

```
lerAlunos1(F):- seen, codsAlunos(Lca),
                see(F), processar(Lca).
processar(Lca):- read(T), tratar1(T, Lca).
tratar1(end_of_file, _):- seen, !.
tratar1(aluno(C, N, I), Lca):- member(C, Lca), !, retract(aluno(C, _, _)),
                               assert(aluno(C, N, I)), processar(Lca).
tratar1(aluno(C, N, I), Lca) :- assert(aluno(C, N, I)), processar(Lca).
```

% 4. - Grafos

```
:- dynamic(depende/2).

instalacao(nero,[driverDVD,driverCD]).
instalacao(photoshop,[nero,windows]).
instalacao(driverDVD,[windows]).
instalacao(driverCD,[]).
instalacao(windows,[]).
instalacao(bluej,[java,windows]).
instalacao(java,[sdk,jbuilder]).
instalacao(jbuilder,[jdbc]).
```

% a)

```
geradependencias :- findall(_,
                           (instalacao(C,Lcod),member(X,Lcod),assert(depende(C,X))),
                            _).
```

% b)

```
produtosIndependentes(Lprod, Lindp):-
        findall(Li,
               (member(C,Lprod),prodInd(C,Li)),
               Lf),
        flatten(Lf,LFinal),
        sort(LFinal,Lindp).

prodInd(Prod,Lind):-
        % poderiamos ter depende(C,_) e findall(...)
        % mas com o setof se o resultado for lista vazia falha,
        % temos o mesmo comportamento
        setof(P,depende(Prod,P),Ld),
        produtosIndependentes(Ld,Lind).

% prodInd(Prod,[Prod]) :- not(depende(Prod,_)).
                        % independente se ninguem depende dele.

/*-------------------------------------------------------------------------*/
```