```
/*-----------------------------------------------------------------*/
/* EXAME DE PPIII 2006/2007 - 2ª. CHAMADA                          */
/*              PROPOSTA DE RESOLUÇÃO                              */
/*-----------------------------------------------------------------*/


% PARTE I
% Conta bits de uma lista do tipo [1,0,1,0,1,1,0]

contaBits([], 0/0):- !.
contaBits([1|T], Z/U):- contaBits(T, Z/Ut), U is Ut + 1.
contaBits([0|T], Z/U):- contaBits(T, Zt/U), Z is Zt + 1.

% converte lista de 0s e 1s para decimal

converteDec1([], 0) :- !.
converteDec1([H|T], Dec) :- length(T, P), Val is H*(2**P),
                            converteDec1(T, Dect), Dec is Val + Dect.

% Dada uma lista de listas determinar a lista de pares Indice/Comp

indiceComp([], []) :- !.
indiceComp(L, Lind) :- indiceCompAux(L, 1, Lind).

indiceCompAux([H], I, [I/C]) :- !, length(H, C).
indiceCompAux([H|T], I, [I/C|Lc]) :- I1 is I+1, length(H, C),
indiceCompAux(T, I1, Lc).


2.

%a) Remover alunos com números entre N1 e N2, com N1 > N2

removerEntre(N1, N2) :- findall(_, ( aluno(N, _,_),
                        N >= N1, N =< N2,
                        retract(aluno(N, _,_)) ), _).

removerEntre1(N1, N2) :- aluno(N, _, _), N >= N1, N =< N2,
                         retract(aluno(N, _,_)), fail.
removerEntre1(_,_).

%b) Calcular lista de pares Curso/MediaGeraldoCurso

cursos(Lc) :- setof(C, Num^M^aluno(Num,C,M), Lc).
cursos([]).

mediaCurso(C, M) :- bagof(N, Num^aluno(Num,C, N), Ln),
                    somaLista(Ln, S), length(Ln, Nn), M is S/Nn.
somaLista([], 0) :- !.
somaLista([H|T], S) :- somaLista(T, S1), S is H + S1.

paresCursoMedia(Lpcm) :- cursos(Lc),
                         setof(C/M,
                              (member(C, Lc), mediaCurso(C, M)), Lpcm).

% OU AINDA
```

```
paresCursoMedia(L) :- findall(C/M, cursoMedia(C; M), L).

cursoMedia(C, M):- bagof(Ma, N^aluno(N, C, Ma), L),
                   length(L, T), sumlist(L, S). M is S/T.
```

%c) Ler alunos de um dado curso a partir de ficheiro

```
/* Ler de ficheiro apenas os alunos de um dado Curso e carregar a BC
usando REPEAT */


inserePorCurso(C, F):- seen, see(F),  % acessFile(F, read)
                       repeat,
                         read(T),
                         (T == end_of_file, seen, !
                         ;
                         tratar(T, C), fail).
tratar(aluno(N, _, _), _) :- aluno(N,_,_), !.
tratar(aluno(N, C, M), C) :- assert(aluno(N, C, M)).
```

/* Versão recursiva; */

```
inserePorCurso1(C, F):- seen, see(F), processar(C).
processar(C):- read(T), tratar1(T, C).
tratar1(end_of_file, _):- seen, !.
tratar1(aluno(N, _, _), C):- aluno(N, _, _), processar(C).
tratar1(aluno(N, C, M), C) :- assert(aluno(N, C, M)), processar(C). % tem
que ser do curso C
tratar1(_,C) :- processar(C).
```

%d) Lista de Alunos com melhor média

```
melhorMedia(M) :- setof(Med, Num^C^aluno(Num, C, Med), Lm), last(M, Lm).
alunosComMelhorMedia(La) :- melhorMedia(M),
                            findall(Num, aluno(Num, _, M), La).

% OU AINDA

melhorAluno(L):- setof(M/La, C^bagof(N, aluno(N, C, M), La), Laux),
                 last(_/L, Laux).
```

% PARTE III

```
% dynamic(viajemDirecta/2).

liga(a, b).
liga(b, c).
liga(a, d).
liga(d, f).
liga(c, a).
liga(f, c).
liga(b, f).
liga(g, a).
liga(a, h).
```

% Criar predicados de 1:N ligações

```
nodos(Ln) :- setof(N, X^(liga(N, X) ; liga(X, N)), Ln).
criaLigacoes :- nodos(Ln), findall(_, (member(N, Ln),
                                       findall(Nd, liga(N, Nd), Ll),
                                  % setof nao dá [] !!
                                       assert(viajemDirecta(N, Ll))), _).
```

% Lista de Cidades que dão acesso a uma cidade Ct

```
acesso(C1, C2) :- liga(C1, C2), !.
acesso(C1, C2) :- liga(C1, Cx), acesso(Cx, C2).

daoAcesso(Ct, L) :- nodos(Ln),
                    setof(C, (member(C, Ln), acesso(C, Ct)), L).

% ou excluindo partidas da propria cidade

daoAcesso1(Ct, L) :- nodos(Ln),
                   setof(C, (member(C, Ln), C \== Ct, acesso(C, Ct)), L).
```

/*_____*/