Universidade do Porto Faculdade de Engenharia FEUP	Faculdade de Engenharia da Universidade do Porto **Mestrado Integrado em Eng. Informática e Computação** **Programação em Lógica** *Época de Recurso – Com Consulta / Duração: 2h30m*		**2007/2008** **3º Ano MIEIC**
Nome:		**Data:**	**11/02/2008**

Nota: Não utilize nenhuma biblioteca do Prolog, excepto a CLPFD, para resolver o exame. Predicados definidos em alíneas anteriores (mesmo que não os implemente) podem ser utilizados em alíneas seguintes.

GRUPO I – Participações em Filmes (4 Val)

O IMDB disponibiliza desde há algum tempo informação sobre personagens, para além de actores e filmes/séries. Suponha que o seguinte conjunto de factos é ilustrativo de uma base de conhecimento, onde é indicado o nome do personagem, o nome do filme/série em que foi representado, e o nome do actor que o interpretou:

```
by(jackRyan, theSumOfAllFears, benAffleck).
by(jackRyan, patriotGames, harrisonFord).
by(frasierCrane, cheers, kelseyGrammer).
by(joeyTribbiani, friends, mattLeBlanc).
by(stephenColbert, dailyShow, stephenColbert).
by(addisonMontgomery, privatePractice, kateWalsh).
by(elektraNatchios, daredevil, jenniferGarner).
by(elektraNatchios, elektra, lauraWard).
by(jackRyan, theHuntForRedOctober, alecBaldwin).
by(jackRyan, clearAndPresentDanger, harrisonFord).
by(frasierCrane, frasier, kelseyGrammer).
by(joeyTribbiani, joey, mattLeBlanc).
by(stephenColbert, theColbertReport, stephenColbert).
by(addisonMontgomery, greysAnatomy, kateWalsh).
by(elektraNatchios, elektra, jenniferGarner).
```

1.1) Construa o predicado ***playedBy(+Name, -List)***, que devolve em *List* uma lista com todos os actores que representaram a personagem identificada por *Name*, contendo uma lista com os respectivos filmes/séries, conforme os exemplos abaixo. Quando o nome do personagem coincide com o nome do actor, deve apresentar 'self' em vez do nome do actor. **(1,5 Val)**

Exemplo:

```
?- playedBy(jackRyan, List).
List = [    benAffleck-[theSumOfAllFears],
            alecBaldwin-[theHuntForRedOctober],
            harrisonFord-[patriotGames, clearAndPresentDanger]]
?- playedBy(stephenColbert, List).
List = [self-[dailyShow, theColbertReport]]
```

1.2) Construa o predicado ***aged(+Character, +Movie)***, que sucede caso a personagem indicada tenha mais do que um actor que a interprete no mesmo filme/série (normalmente indicativo de mais do que uma fase da sua vida). **(1 Val)**

Exemplo:

```
?- aged(elektraNatchios, daredevil).
No.
?- aged(elektraNatchios, elektra).
Yes.
```

1.3) Construa o predicado ***mostPopular(+ Exclude, -List, -NMovies)***, que devolve em *List* os nomes dos personagens que entraram em mais filmes diferentes (esta lista só terá mais que um valor em caso de empate), e em *NMovies* o número de filmes em que entraram. Devem ser excluídos os personagens indicados na lista *Exclude*. **(1,5 Val)**

Exemplo:

```
?- mostPopular([], List, NMovies).
List = [jackRyan], NMovies = 4
?- mostPopular([jackRyan], List, Nmovies).
List = [frasierCrane, joeyTribbiani, stephenColbert, addisonMontgomery,
electraNatchios], NMovies = 2
```

GRUPO II – Programação em Prolog (3,5 Val)

2.1) Escreva um predicado ***tabuleiro(+N1, +N2)*** em Prolog que, dadas duas medidas *N1* e *N2* (ambas ímpares), desenhe um tabuleiro composto por um quadrado exterior de lado *N1* e um losango interior de altura *N2* (com *N2 <= N1*), como apresentado nos exemplos**. (1,5 Val.)**

Exemplos:

```
?-tabuleiro(1,1).
*

?-tabuleiro(5,1).
*****
*   *
* * *
*   *
*****

?-tabuleiro(7,5).
*******
*  *  *
* * * *
**   **
* * * *
*  *  *
*******
```

```
?- tabuleiro(9,5).
*********
*       *
*   *   *
*  * *  *
* *   * *
*  * *  *
*   *   *
*       *
*********

?- tabuleiro(9,9).
*********
*  * *  *
* *   * *
**     **
*       *
**     **
* *   * *
*  * *  *
*********
```

```
?-tabuleiro(10,5).
Impossível! N1 Par!

?-tabuleiro(9,4).
Impossível! N2 Par!

?-tabuleiro(5,7).
Impossível! N2>N1!

?-tabuleiro(8,10).
Impossível! N1 Par! N2>N1!
```

2.2) Suponha o seguinte excerto de código Prolog.

```
q(a).
q(b).
r(1).
r(2).
r(3).

s(Y):- r(Y),!.

p1(X,Y):- q(X),r(Y).
p2(X,Y):- r(Y),q(X).
p3(X,Y):- q(X),!,r(Y).
p4(X,Y):- q(X),r(Y),!.
p5(X,Y):- q(X),s(Y).
```

Diga justificando todas as respostas fornecidas pelo Prolog às seguintes "queries": **(1 Val)**

```
a) ?- p1(X,Y).
b) ?- p2(X,Y).
c) ?- p3(X,Y).
d) ?- p4(X,Y).
e) ?- p5(X,Y).
```

2.3) Indique quais as respostas do Prolog às seguintes questões? Justifique. **(1 Val)**

```
a) ?- N = 3, N is N+1.
b) ?- N == 5, N2 is N+1.
c) ?- [A|B] = [1,2,3,4].
d) ?- [A,B|C] = [1,2,3|[]].
e) ?- a = A, A == a.
f) ?- 1+3 = 3+1.
g) ?- 10 =:= 2*5.
h) ?- arg(A,a(a,b,c),b).
```

GRUPO III – The Da Vinci Code (5,5 Val)

Considere um codex, como o que é mostrado na figura (e que pode ser visto em O Código Da Vinci), representado por uma lista de listas, tal como mostrado abaixo. Considere que todas as listas possuem o mesmo tamanho.

```
[ [a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
  [p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o]
  [p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o]
  [l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k]
  [e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d]
 ]
```

3.1) Construa o predicado ***shift(+List, +Rotations, -NewList)***, que roda os elementos de uma lista *List* *Rotations* vezes (o primeiro elemento fica em último no caso de uma rotação) e devolve o resultado em *NewList*. Caso *Rotations* seja negativo deve ser considerado como um shift no sentido inverso. **(1 Val)**

Exemplo:

```
?- shift( [2, 4, 6, 8], 1, NewList).
NewList = [4, 6, 8, 2]
?- shift( [2, 4, 6, 8], -1, NewList).
NewList = [8, 2, 4, 6]
```

3.2) Construa o predicado ***rotate(+List, +Rotations, +From, -NewList),*** que dada uma list *List* com a configuração inicial do codex, devolva em *NewList* a lista rodada de *Rotations* vezes, a partir da posição *From* (considere a primeira posição como posição 1). O predicado deverá falhar caso tente aplicar uma rotação a partir de uma posição anterior à primeira ou posterior à última. **(1 Val)**

Exemplo:

```
?- rotate([ [3,6,8], [5,0,7], [2,1,4] ], 1, 2, NewList).
NewList = [ [3,6,8], [0,7,5], [1,4,2] ]
?- rotate([ [3,6,8], [5,0,7], [2,1,4] ], 2, 3, NewList).
NewList = [ [3,6,8], [5,0,7], [4,2,1] ]
```

3.3) Construa o predicado ***break(+List, +Code)***, que sucede caso *List* esteja alinhada com *Code* como solução, e falhe caso contrário. O predicado deve ainda imprimir mensagens de erro caso *Code* tenha tamanho diferente do número de elementos do codex. (**1 Val**)

Exemplo:

```
?- break([ [3,6,8], [5,0,7], [2,1,4] ], [3,5,2]).
yes
?- break([ [3,6,8], [5,0,7], [2,1,4] ], [6,0,1]).
no
?- break([ [3,6,8], [5,0,7], [2,1,4] ], [4,6,0,1]).
O código tem tamanho diferente do codex
no
```

3.4) Construa o predicado ***operator(+Initial, -Final),*** que devolve uma a uma todas as possibilidades de jogo (rotações entre 1 e o número de elementos das listas, a partir de cada uma das posições). **(1 Val)**

Exemplo:

```
?- operator([[3,6,8],[5,0,7],[2,1,4]], Final).
Final = [[6,8,3],[0,7,5],[1,4,2]] ;
Final = [[8,3,6],[7,5,0],[4,2,1]] ;
...
Final = [[3,6,8],[5,0,7],[4,2,1]] ;
no.
```

3.5) Construa o predicado ***solve(+List, +FinalList, -Operat),*** que devolve em *Operat* o conjunto de estados por onde passa o codex para ser resolvido, evitando estados repetidos. *FinalList* contém o código, neste caso, representando o primeiro elemento de cada lista. **(1.5 Val)**

Exemplo:

```
?- solve([[3,6,8],[5,0,7],[2,1,4]], [6,0,4], Operat).
Operat = [ [[6,8,3],[0,7,5],[1,4,2]] ]
?- solve([[3,6,8],[5,0,7],[2,1,4]], [6,5,1], Operat).
Operat = [ [[6,8,3],[0,7,5],[1,4,2]], [[6,8,3],[5,0,7],[2,1,4]] ]
```

GRUPO IV – Programação em Lógica com Restrições (7 val)

Nota: Não esquecer de utilizar a biblioteca clpfd e que as variáveis utilizadas são inteiras (domínio finito).

4.1) Utilizando Programação em Lógica com Restrições, construa um programa capaz de determinar 3 números tais que o produto de quaisquer dois deles, adicionado ao terceiro, dê um quadrado perfeito.**(1 Val)**

4.2) Numa visita a Portugal, os ilustres visitantes da FIFA, tiveram oportunidade de visitar sequencialmente as instalações de cinco clubes de futebol portugueses (F.C.Porto, Sp.Braga, Sporting, Benfica e Belenenses), cada um fundado numa data distinta (1893, 1904, 1906, 1919 e 1921). Sabendo que:

- O Benfica é 2 anos mais antigo do que o Sporting;
- O 3º clube visitado foi fundado em 1893;
- O Sp. Braga foi visitado imediatamente antes do que o clube que foi fundado em 1906;
- A visita ao F.C.Porto (que não foi o último clube a ser fundado) foi anterior à visita ao Benfica que por sua vez foi visitado antes do que o clube fundado em 1919.

Pretende-se determinar a ordem de visita aos cinco clubes e a data de fundação de cada um. Para tal construa um programa utilizando PLR. **(1,5 Val)**

4.3) Como colocar os 16 peões num tabuleiro de xadrez (8x8 quadrículas) de modo a que não existam 3 em qualquer linha (horizontal ou vertical) considerada (i.e. existam 2 em cada)? Para resolver este problema construa um programa em PLR capaz de determinar a colocação de N*2 peões num tabuleiro com N*N quadriculas respeitando a regra enunciada. **(1,5 Val)**

4.4) Pretende-se obter o horário para uma conferência com diversos artigos que vão ser apresentados oralmente, participantes e várias sessões paralelas. Alguns dos participantes vão apresentar artigos na conferência e os artigos têm diversos temas (i.e. cada artigo tem pelo menos um tema podendo ter vários).
Os artigos devem ser agrupados em sessões temáticas com o mesmo número de artigos (suponha como simplificação que o número de artigos é múltiplo do número de sessões). Numa determinada sessão temática, todos os artigos devem ter pelo menos um tema em comum. Todos os autores de um artigo devem poder estar presentes na respectiva sessão. Esta é uma restrição rígida. Cada participante assiste a sessões completas e não pode estar em duas sessões ao mesmo tempo.
Como os "palestrantes" deste tipo de conferências conhecem-se bem, têm também algumas restrições/incompatibilidades relativas a estarem na mesma sala com outros participantes! Esta é uma restrição flexível que se pretende cumprir dentro das possibilidades.
Adicionalmente, cada participante está interessado em assistir à apresentação de diversos artigos (tem preferências sobre isto). Para além de cumprir as restantes restrições, pretende-se minimizar o número de artigos a que os participantes não assistem mas estavam interessados em assistir e minimizar as incompatibilidades não respeitadas.
Construa um programa em PLR para resolver este problema, determinando em que sessão é apresentado cada artigo e quais as sessões em que cada participante vai participar. Suponha que os dados são apresentados do modo indicado em seguida: **(3,0 Val).**

```
sessoes(4, 2).  % 4 Sessões no total, 2 sessões paralelas
areas([ia, rob, es, bd]).  %4 Áreas temáticas
participantes([paulo,antonio,joao,pedro,ana,barbara,carla,carlos,diana,filipe]).
artigo(1, [paulo,antonio], [ia, rob]).     % artigo 1 tem como autores o Paulo e o
                                             António e áreas IA e Robótica
artigo(2, [paulo,joao], [ia, rob, es, bd]). %artigo 2 do Paulo/João foca 4 áreas
artigo(3, [pedro,joao], [ia, rob]).
artigo(4, [pedro, ana, barbara], [bd]).
artigo(5, [ana, carla, carlos], [es, bd]).
artigo(6, [carla, carlos, diana], [es, bd]).
artigo(7, [filipe], [ia, es]).
artigo(8, [diana, filipe], [es, bd]).
incompatibilidade(diana, paulo).  % Diana não quer estar na mesma sessão do Paulo.
incompatibilidade(paulo, diana).  % Paulo não quer estar na mesma sessão do Diana.
incompatibilidade(ana, pedro).
incompatibilidade(carla, filipe).
preferencias(paulo, [3,4]).      % Paulo pretende assistir aos artigos 3 e 4 para
                                   além dos 1 e 2, dos quais é autor
preferencias(joao, []).
preferencias(antonio, [2,3]).
preferencias(ana, [6]).
preferencias(barbara, [6,7,8]).
```

BOM TRABALHO!