Prova com consulta. Duração: 2h30m. Exame da Época Normal

Notas:
a) Responda a cada questão numa folha de exame separada (5 questões = 5 folhas).
b) No Grupo I use apenas Prolog Standard, e no Grupo II use também a biblioteca *CLPFD* do SICStus Prolog.
c) Predicados solicitados em alíneas anteriores podem ser utilizados em alíneas seguintes, mesmo que não os tenha implementado.

GRUPO I – Programação em Prolog

1. [**5 valores**] Considere um jogo em que cada jogador escolhe uma personagem que vai usar no jogo. As personagens têm características/habilidades diferentes, e cada jogador tem a sua preferência sobre que personagem gosta mais: predicado *pref(Player,Character)*. São guardadas estatísticas sobre o desempenho de cada jogador/personagem: predicado *stat(Player,Character,Wins,Losses)*. Uma parte da base de conhecimento poderá ser a seguinte:

```
stat(ana, thresh, 1000, 200).
stat(ana, evelyn, 150, 200).
stat(joao, ezreal, 300, 40).
stat(joao, wukong, 100, 40).
stat(carlos, nidalee, 56, 60).
stat(carlos, thresh, 70, 6).
```

```
pref(ana, thresh).
pref(joao, wukong).
pref(carlos, nidalee).
```

a) Implemente o predicado *samePref(?J1,?J2)*, que sucede com pares de jogadores *J1* e *J2* que preferem a mesma personagem.

b) Implemente o predicado *mostWinsWith(+Char,-J,-Wins)*, que unifica *J* com o jogador que tem mais vitórias com a personagem *Char* e *Wins* com o número de vitorias que obteve.

c) Implemente o predicado *rightChoice(+J)*, que verifica se o rácio vitórias/derrotas do jogador *J* com a sua personagem favorita é o maior de entre as personagens com que já jogou.

d) Implemente o predicado *teamOfGoodChoices(+Team)*, que verifica se todos os jogadores da lista *Team* fazem a escolha certa da sua personagem favorita.

e) Implemente o predicado *teamPrefs(+Team,-Characters)*, que constrói a lista *Characters* com a personagem preferida por cada jogador que está na lista *Team*, pela mesma ordem.

f) Implemente o predicado *differentPrefs(+Team)*, que verifica se todos os jogadores da lista *Team* preferem personagens diferentes.

2. [**4 valores**] A Direção Geral de Viação da Mieiclândia mantém uma base de dados dos veículos e condutores em Prolog, com factos dos seguintes tipos:

vehicle(Marca, Modelo, Matricula, DataRegisto)
stamp(Matricula, DataInicio, DataFim)
person(NumeroCartaConducao, Nome, DataNascimento)

Um veículo está autorizado a circular se o selo estiver em dia. Considere que as datas são representadas por *Dia-Mes-Ano* e existem os seguintes predicados já implementados:

```
today(Data)              % unifica Data com a data atual
dif_days(Data1,Data2,Dif)  % unifica Dif com Data2-Data1, em dias
```

a) Escreva o predicado *n_vehicles(-Num)*, que obtém em *Num* o número total de veículos registados.

b) Escreva o predicado *authorized_vehicles(-List)*, que obtém em *List* as matrículas de todos os veículos autorizados a circular à data atual.

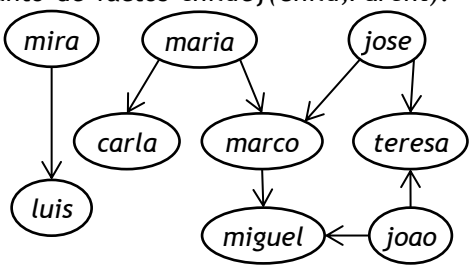
c) Considere o código que se segue:

```
a(X):- person(X,_,B), !, today(H), dif_days(B,H,D), 18 >= D/365.
a(_).
```

c1) Diga o que faz o predicado *a*/1. Que nome atribuiria a este predicado?

c2) O *cut* no predicado *a*/1 é verde ou vermelho? Justifique.

3. [**4 valores**] Uma árvore genealógica pode ser representada por um grafo orientado: cada nó representa um indivíduo, cuja filiação (pai e mãe biológicos) é representada por duas ligações. Esta relação está representada por um conjunto de factos *childOf(Child,Parent)*. Para o exemplo apresentado, teremos:




```
childOf(mira,luis).     childOf(maria,carla).
childOf(maria,marco).   childOf(jose,marco).
childOf(jose,teresa).   childOf(marco,miguel).
childOf(joao,teresa).   childOf(joao,miguel).
```

Pretende-se detetar se dois indivíduos são parentes e qual o seu grau de parentesco, medido através da soma das distâncias de cada um ao antepassado comum mais próximo.

a) Explique o que faz o predicado *c/3* seguinte:

```
c(X,Y,D) :- c(X,Y,D,[]).
c(X,Y,1,_) :- childOf(X,Y).
c(X,Y,D,L) :-
        childOf(X,Z), \+ member(Z,L),
        c(Z,Y,D1,[Z|L]), D is D1+1.
```

b) Com base no predicado apresentado, construa um predicado *relative(I1,I2,G)* que determine se *I1* e *I2* são parentes e que obtenha o seu grau de parentesco *G*. No caso de não serem parentes, o predicado deve suceder com *G*=0.

Prova com consulta. Duração: 2h30m. Exame da Época Normal

GRUPO II – Programação em Lógica com Restrições

4. [4 valores]

a) No contexto de resolução de um problema usando programação em lógica com restrições, depara-se com as seguintes três abordagens alternativas para restringir os valores de duas variáveis. Indique, justificando, qual das alternativas escolheria.

```
…                                  |  …                                  |  …
a_b(5, 7).                         |  constrain2(A,B) :-                 |  constrain3(A,B) :-
a_b(7, 11).                        |     (A #= 5 #/\ B #= 7) #\/         |     (A = 5, B = 7) ;
constrain1(A,B) :- a_b(A,B).       |     (A #= 7 #/\ B #= 11).           |     (A = 7, B = 11).
```

b) Implemente um programa em PLR que permita obter sequências de 6 dígitos distintos entre 1 e 7 onde não podem existir dois dígitos pares consecutivos nem dois dígitos ímpares consecutivos. Adicionalmente, cada subsequência de dois dígitos não pode ter dígitos consecutivos (e.g., as subsequências [3, 4] e [4, 3] são inválidas). Exemplo:

```
?- sequence(L).
L = [2, 7, 4, 1, 6, 3] ? ;
L = [5, 2, 7, 4, 1, 6] ? ;
no
```

Note que devem ser descartadas soluções simétricas (e.g., L = [3, 6, 1, 4, 7, 2] não deve ser indicada como solução).

c) Generalize o problema da alínea anterior para sequências de dimensão *N* de elementos entre 1 e *N*+1, considerando ainda que as restrições indicadas (paridade e subsequência) se aplicam também entre o último e o primeiro elementos, formando assim uma sequência circular. Exemplo:

```
?- sequence(8, L).
L = [1, 4, 7, 2, 5, 8, 3, 6]
```

5. [3 valores] No problema 1 foi-lhe apresentado um jogo para o qual temos informação sobre a personagem preferida por cada jogador – *pref/2* – e sobre o seu desempenho com cada personagem – *stat/4*. Considere que se pretende constituir uma equipa forte de *N* jogadores, tendo em conta as suas preferências mas também o seu desempenho:

- Se a escolha do jogador for certa (ver *rightChoice/1* da alínea *c* do problema 1), então a sua preferência tem que ser contemplada;
- A utilização das diferentes personagens não pode diferir em mais de 1 unidade: por exemplo, só pode haver dois jogadores com a mesma personagem se todas as outras personagens já estiverem a ser utilizadas (note que a conjugação desta restrição com a anterior pode inviabilizar a existência de solução);
- Uma solução é tanto melhor quanto maior for a soma dos rácios 100*(vitórias//derrotas) de todos os pares jogador/personagem na solução.

Esboce o mais detalhadamente possível um programa em PLR que, dada uma lista de jogadores, permita obter a solução ótima de atribuições de personagens a jogadores, observando as restrições indicadas. As personagens existentes estão no seguinte facto:

```
charsMap([1-thresh, 2-evelyn, 3-ezreal, 4-wukong, 5-nidalee]).
```

Exemplos (observando a base de dados do problema 1):

```
?- team([ana, joao, carlos], T).
T = [thresh, ezreal, nidalee]

?- team([joao, carlos], T).
T = [ezreal,thresh]
```

```
?- team([ana, ana], T).
no

?- team([joao, joao], T).
T = [ezreal, wukong]
```