Universidade do Porto Faculdade de Engenharia **FEUP**	Faculdade de Engenharia da Universidade do Porto **Mestrado Integrado em Eng. Informática e Computação** **Mestrado Integrado Eng. Electrotécnica e de Computadores** **Programação em Lógica** *Época Normal – Com Consulta / Duração: 2h30m*	**2011/2012** **3º Ano MIEIC** **4º/5º Ano MIEEC** **17/01/2012**

Notas:

a) No Grupo I use apenas Prolog Standard, e no Grupo II use também a biblioteca CLPFD do SICStus Prolog.

b) Predicados solicitados em alíneas anteriores podem ser utilizados em alíneas seguintes, mesmo que não os tenha implementado.

c) Responda a cada questão (agrupamento de alíneas) numa folha separada: N questões = N folhas.

GRUPO I – Programação em Prolog (13 valores)

1. [5 valores]
Considere a seguinte base de dados com informação sobre diversos países:

```
country(portugal).
country(spain).
country(france).
country(germany).
country(usa).
country(brasil).
country(china).
…
```

```
place(portugal, europe).
place(spain, europe).
place(france, europe).
place(germany, europe).
place(usa, america).
place(brasil, america).
place(china, asia).
…
```

```
border(portugal, spain).
border(france, spain).
border(france, germany).
…
continent(europe).
continent(america).
continent(asia).
…
```

```
popul(portugal, 10).
popul(spain, 48).
popul(france, 52).
popul(germany, 82).
popul(usa, 235).
popul(brasil, 155).
popul(china, 1100).
…
```

a) Implemente o predicado ***next_to(+P1, ?P2)***, que unifica ***P2*** com um país fronteiriço a ***P1***.

b) Implemente o predicado ***big_country(?P)***, que unifica ***P*** com um país com mais de 50 milhões de habitantes.

c) Implemente o predicado ***most_populated(?P)***, que unifica ***P*** com o país mais populoso.

d) Implemente o predicado ***two_most_populated(?P1, ?P2)***, que unifica ***P1*** e ***P2*** com os dois países mais populosos.

e) Implemente o predicado ***not_biggest_within_continent(?P)***, que sucede se ***P*** não é o maior país do seu continente.

f) Implemente o predicado ***relatively_close(?P1, ?P2)***, que sucede se ***P1*** e ***P2*** são fronteiriços ou se basta atravessar um outro país para chegar de ***P1*** a ***P2***.

g) Uma tentativa de implementação do predicado ***n_borders(?P1, ?P2, +N)***, em que ***N*** é o número de fronteiras que é necessário transpor para chegar de ***P1*** a ***P2***, resultou no seguinte:

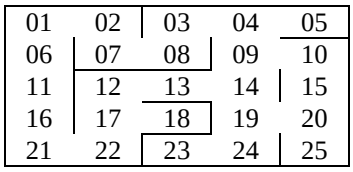
```
n_borders(P, P, 0) :- country(P).
n_borders(P1, P2, N) :-
      next_to(P1, Pi), N1 is N-1, n_borders(Pi, P2, N1).
```

Para uma qualquer pergunta utilizando o predicado ***n_borders***, a computação termina? Se sim, apresente a árvore de pesquisa completa para a pergunta ***?- n_borders(portugal, P2, 2)***, apenas com os factos do enunciado. Se não, altere o predicado de modo a obter sempre computações finitas.

h) Pretende-se obter pares de países grandes que estejam relativamente próximos um do outro. Para tal, definiu-se a seguinte pergunta: ***?- relatively_close(P1, P2), big_country(P1), big_country(P2).*** Comente sobre o esforço de pesquisa desta pergunta e sugira uma formulação alternativa.

2. [5 valores]

Considere o labirinto representado na figura. Este é representado em Prolog através de uma lista de termos ***con(A,B)***, em que A<B, indicando que é possível ir de A para B: `[con(1,2), con(1,6), ...]`



a) Pretende-se um programa capaz de construir iterativamente um labirinto deste tipo. Para tal, é necessário um predicado ***newConnection(+Lab, +A, +B, -NewLab)***, que acrescenta ao labirinto ***Lab*** uma ligação de ***A*** para ***B***, resultando daí ***NewLab***. Esta lista deve estar sempre ordenada pela origem (***A***) de cada ligação, e depois pelo destino (***B***). Considere a seguinte resolução:

```
newConnection([], A, B, [con(A,B)]).
newConnection([con(X,Y)|Z], A, B, [con(A,B)|[con(X,Y)|Z]]) :- A<X, !.
newConnection([con(A,Y)|Z], A, B, [con(A,B)|[con(A,Y)|Z]]) :- B<Y, !.
newConnection([XY|Z], A, B, [XY|W]) :- newConnection(Z, A, B, W).
```

Os *cuts* utilizados são verdes ou vermelhos? Justifique.

b) Reimplemente o predicado da alínea anterior sem utilizar *cuts*, e verificando que a origem é menor do que o destino (A<B). Caso contrário, o predicado deverá inverter estes elementos (e.g., se são passados A=4 e B=3, deverá ser adicionada a ligação ***con(3,4)*** e não ***con(4,3)***). O predicado deve falhar no caso de A=B.

c) Escreva o predicado ***limitedPath(+Lab, +C1, +C2, -Path)*** que encontra no labirinto ***Lab***, por *backtracking*, caminhos ***Path*** válidos entre duas casas ***C1*** e ***C2*** dadas, em que nunca se vai de uma casa maior para uma menor (note que os elementos ***con(A,B)*** de ***Lab*** respeitam a condição A<B).

d) Escreva o predicado ***path(+Lab, +C1, +C2, -Path)*** que encontre no labirinto ***Lab***, por *backtracking*, caminhos ***Path*** válidos entre duas casas ***C1*** e ***C2*** dadas, sem qualquer outra condicionante.
(*Nota*: tenha em conta a pesquisa em profundidade do Prolog, que pode originar facilmente computações infinitas…)

3. [3 valores]

Suponha a seguinte base de conhecimento em Prolog sobre músicas, os artistas/grupos que as cantam e o número de visualizações que o vídeo respetivo tem no YouTube.

```
% song(NameOfTheSong, Interpreter, MillionsOfViews).
song('Sexy and I know it', 'LMFAO', 350).
song('Party Rock Anthem', 'LMFAO', 350).
song('Champagne Showers', 'LMFAO', 75).
song('Friday', 'Rebecca Black', 18).
song('My moment', 'Rebecca Black', 32).
song('Baby', 'Justin Bieber', 690).
song('YMCA', 'Village People', 80).
song('Macho Man', 'Village People', 10).
song('Tenacious D', 'Kickapoo', 30).
```

a) Escreva o predicado ***biggestHits(+Artist, -List)*** que, dado um determinado artista, devolve o(s) seu(s) maior(es) *hit*(s), ou seja, a(s) música(s) com maior número de visualizações. Por exemplo:

```
?- biggestHits('LMFAO', List).
List = ['Sexy and I know it', 'Party Rock Anthem'].
?- biggestHits('Rebecca Black', List).
List = ['My moment'].
```

b) Escreva o predicado ***topSongs(+Value, -List)*** que obtenha a lista de todos os artistas e suas músicas que tenham mais visualizações do que o valor passado em ***Value***. Por exemplo:

```
?- topSongs(75, List).
List = ['Justin Bieber'-['Baby'], 'LMFAO'-['Sexy and I know it', 'Party
Rock Anthem'], 'Village People'-['YMCA']].
?- topSongs(200, List).
List = ['Justin Bieber'-['Baby'], 'LMFAO'-['Sexy and I know it', 'Party
Rock Anthem']].
```

GRUPO II – Programação em Lógica com Restrições (7 valores)

Na resposta às seguintes questões, utilize a biblioteca *CLPFD* do SICStus Prolog.

4. [4 valores]

a) Pretende-se preencher um quadrado de 3x3 com algarismos entre um e nove. Cada algarismo não pode aparecer mais do que uma vez. A soma dos algarismos colocados em cada uma das linhas deverá ser 15. Construa um predicado que permita obter a solução. Exemplo:

```
?- square(Vars).
Vars = [1,5,9,2,6,7,3,4,8] ?
```

1	5	9
2	6	7
3	4	8

b) Implemente um predicado que resolva uma variante do problema anterior, em que podem haver N quadrados de 3x3 dispostos verticalmente, tendo em conta as seguintes restrições:

- Cada célula pode tomar um valor entre um e nove, como anteriormente;
- A soma dos algarismos de cada uma das linhas deverá ser 15, como anteriormente;
- Cada algarismo pode aparecer repetido N vezes (em que N é o número de quadrados 3x3);
- Em cada linha: os algarismos devem estar ordenados do maior para o mais pequeno e não podem haver algarismos repetidos;
- O primeiro algarismo de cada duas linhas consecutivas não pode ser igual.

Exemplo:

```
?- squares(Vars, 2).
Vars = [9,5,1,8,6,1,9,4,2,7,6,2,8,4,3,7,5,3] ?
```

9	5	1
8	6	1
9	4	2
7	6	2
8	4	3
7	5	3

c) Indique o conjunto de *flags* a passar como argumento ao *labeling* tendo em vista maximizar a eficiência de execução para o problema da alínea b). Justifique, e caso passe mais do que uma *flag* indique qual é a mais crítica.

d) Implemente um predicado auxiliar para o problema b) que force a que o número de linhas que contêm exatamente dois números múltiplos de quatro esteja entre um e três. Indique como alteraria o predicado da alínea b).

5. [3 valores]

Uma máquina produz 4 tipos de peças: 0, 1, 2 e 3. A primeira peça a fabricar não pode ser do tipo 0. A máquina, depois de produzir uma peça, está restringida relativamente à peça que pode produzir a seguir. O custo de produção de uma peça também depende da peça que foi produzida anteriormente. Estas relações estão sumarizadas na seguinte tabela (e.g. depois de ser produzida uma peça do tipo 2 pode ser produzida uma do tipo 0 por 1€ ou uma do tipo 1 por 4€):

Peça anterior \ *Peça seguinte*	*0*	*1*	*2*	*3*
0		**3€**	**3€**	**2€**
1				**1€**
2	**1€**	**4€**		
3		**5€**		**2€**

Implemente um predicado que, dado o número exato de peças a produzir (quantidade por tipo de peça), indique a sequência de peças que minimiza o custo total de produção (assuma que a primeira peça tem custo zero).

Exemplo de uma possível execução em que devem ser fabricadas 2 peças de cada tipo:

```
?- seq(Sequence, 2, Cost).
Cost = 15,
Sequence = [2,0,2,0,1,3,1,3] ?
```