Universidade do Porto Faculdade de Engenharia FEUP	Faculdade de Engenharia da Universidade do Porto **Mestrado Integrado Engenharia Informática e Computação** **Mestrado Integrado Eng. Electrotécnica e de Computadores** **Programação em Lógica** *Época da Época Normal – Com Consulta / Duração: 3h00m*		**2010/2011** **3º Ano MIEIC/ 4º Ano MIEEC**
Nome:		**Data:**	**18/01/2010**

Nota: Utilize exclusivamente Prolog Standard no grupo I. No grupo II utilize Prolog Standard e a biblioteca CLPFD do SICStus Prolog. Responda em folhas separadas aos seguintes grupos de perguntas: Folha 1: Grupo 1, Pergunta 1; Folha 2: Grupo 1, Perguntas 2, 3 e 4; Folha 3: Grupo 1, Pergunta 5; Folha 4: Grupo II, Perguntas 6 e 7; Folha 5: Grupo II, Perguntas 8 e 9.

GRUPO I – Programação em Prolog (13 val.)

1) A "NET Jogos", loja on-line de venda de videojogos, atendendo à actual taxa de crescimento do seu volume de negócio, pretende desenvolver um sistema de gestão de encomendas utilizando Prolog. (**5 Val**).

1.1) Criar um ficheiro de dados para o sistema de gestão de encomendas (**1 Val**):

a) Comece por criar o predicado `cliente/3`. O primeiro argumento é o código do cliente, o segundo é o nome e o último é a morada (cidade apenas). Crie 5 clientes.
b) De seguida crie o catálogo da loja. Para tal crie o predicado que define cada artigo disponível para venda. Deve ter também três argumentos: o código, o nome do artigo (jogo) e a plataforma (PC/PS3/XBOX). Crie 10 artigos.
c) Finalmente crie o registo de inventário para cada artigo definido na alínea anterior. Este registo tem dois argumentos: o código do artigo e a quantidade disponível em stock.

1.2) Para a base de factos definida na alínea anterior, como perguntaria na consola: (**0.75 Val**):

a) Quais os clientes da cidade do Porto?
b) Qual a morada do cliente 1?
c) Quais os jogos disponíveis em stock para PC?

1.3) Escreva uma regra `stock/0` que escreve para a consola um relatório de existências em stock. Deve ser apresentado o nome do artigo, a plataforma e a quantidade em stock (**1 Val**).

Exemplo:

```
|?- stock.
'FIFIA 2012'     'PC'       10
'FIFIA 2012'     'PS3'      9
'FIFIA 2012'     'XBOX'     0
'Bananoid 3'     'PC'       2
'Bananoid 3'     'PS3'      2
'Bananoid 3'     'XBOX'     2
```

1.4) Escreva um predicado `validar_encomenda/4` que verifica se a encomenda efectuada por um cliente é válida. Os argumentos devem ser o cliente, o artigo, a plataforma e a quantidade. O predicado deve suceder se o cliente é um cliente conhecido , existe o artigo em stock e se a quantidade encomendada for menor que o nível de stock. (**1 Val**)

Exemplo:

```
|?- validar_encomenda('Joaquim','FIFIA 2012','PC',2).
yes
```

1.5) Suponha agora que é dada uma lista com os limites mínimos de stock de cada artigo (`Lista_Limites`). Construa o predicado `barras_vert(+Lista_Existencias,+Lista_Limites)`, que dada uma lista de inteiros correspondente às existências em stock e a lista de limites mínimos de cada artigo imprima um gráfico de barras verticais. Cada posição da lista de existências representa o código do artigo e o valor nessa

posição é a quantidade disponível em stock. Por exemplo a lista `[6,4,5,2,5,3]` indica que existem 6 unidades em stock do artigo com código 1, 4 unidades do artigo 2, 5 unidades do artigo 3, etc. Na impressão deve ser usado um 'X' para valores de stock acima do limite mínimo, um 'o' para valores de stock dentro do limite mínimo e um '.' para unidades em falta do artigo, tal como ilustrado no exemplo (**1.25 Val**).
Exemplo:

```
?-
barras_vert([6,4,5,2,5,0], [4,4,3,5,2,2]).
6 X
5 X   X . X
4 o o X . X
3 o o o . X
2 o o o o o .
1 o o o o o .
0 o o o o o o
```

2) Represente cada uma das seguintes frases em Prolog. (**1 Val**).

a) o joão estuda pl.
b) o número de habitantes de França é 50 milhões.
c) a Alemanha é um país rico.
d) alguém escreveu A Mensagem.
e) todos os humanos são mortais.
f) o jorge leva o guarda-chuva se chover.
g) se te portares mal não vês televisão.
h) os seguranças têm mais de 1,80m

3) Quais dos seguintes pares de termos unificam? Onde for relevante, indique as instanciações das variáveis que conduzem à unificação. (**1 Val**).

a) `[a,b|X]=[A,B,c]`
b) `[a,b]=[b,a]`
c) `[a|[b,c]]=[a,b,c]`
d) `[a,[b,c]]=[a,b,c]`
e) `[a,X]=[X,b]`
f) `[a|[]]=[X]`
g) `[a,b,X,c]=[A,B,Y]`
h) `[H|T]=[[a,b],[c,d]]`
i) `[[X],Y]=[a,b]`
j) `1+1 = 2`

4) Escreva um predicado `extrai_pos_par/2` que gere uma segunda lista constituída pelos elementos nas posições pares numa primeira lista passada como argumento: (**2,5 Val**).

```
Exemplo:
|?-extrai_pos_par([4,5,8,9,23,56,75],Result).
Result=[5,9,56]
```

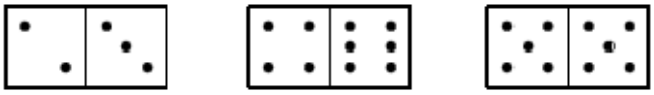
a) Escreva o predicado sem usar cuts.
b) Indique que alterações faria ao código para o optimizar utilizando cuts.
c) Implemente um novo predicado `extrai_geral/3` que pretende generalizar o predicado que escreveu. O 3º argumento deverá ser o nome de um predicado de aridade 1 que será utilizado para filtrar os elementos de interesse. Exemplo:

```
?- extrai_geral([1,2,3,6,8,2], Res, pequeno)
Res = [1,2,3,2]
```

Em que:

```
pequeno(X):- X < 5.
```

5) Domino (3.5 Val)
O conhecido jogo "Domino" é jogado da seguinte forma: cada participante começa com um determinado número de pedras. Cada pedra tem dois números, representados por um conjunto de pontos como se vê na figura seguinte:



No inicio do jogo é colocada uma pedra na mesa e cada jogador na sua vez de jogar deve adicionar uma das suas pedras às pedras já na mesa ou passar a vez. O jogador que primeiro ficar sem pedras vence. Uma pedra apenas pode ser adicionada se na mesa estiver uma pedra com um lado livre e o número que estiver nesse lado coincidir com o número num dos lados da pedra que se pretende adicionar à mesa. Uma pedra com dois números diferentes tem dois lados livres. Uma pedra com dois números iguais tem três lados livres. Quando uma pedra é colocada com um dos seus lados livres em contacto com o lado livre de uma pedra na mesa, esses dois lados passam a ser considerados ocupados.
Iremos considerar a variante chamada "Domino Solitário": é dado um conjunto de pedras e o jogador tenta acabar o jogo sozinho. Assim sendo, inicialmente, escolhe uma das suas pedras para colocar na mesa e depois tenta ir colocando pedra a pedra até não ter mais nenhuma.
Escreva o predicado `domino/1` que, dado um conjunto de pedras sob a forma de factos `pedra/2`, retorna a ordem das pedras a serem colocadas sucessivamente sobre a mesa, respeitando as regras do jogo. Por exemplo, com os factos:

```
pedra(2,2).
pedra(4,6).
pedra(1,2).
pedra(2,4).
pedra(6,2).
```

a seguinte saída estaria correcta:

```
|?- domino(Pedras).
Pedras = [pedra(2,2),pedra(1,2),pedra(2,4),pedra(4,6),pedra(6,2)]
```

NOTAS:
a) pode assumir que não há pedras repetidas.
b) a lista solução pode representar mais do que uma configuração das pedras na mesa, isto é: **a lista representa apenas a ordem pela qual as peças são colocadas, não interessando onde.** Na versão solitária de dominó não interessa qual a pedra que se coloca em primeiro lugar na mesa: a configuração pode ser diferente, mas o resultado final (poder usar as pedras todas ou não) não depende da primeira escolha.

GRUPO II – Programação em Lógica com Restrições (7.0 val)

6) Escreva um predicado em Prolog com restrições para determinar o valor de três números respeitando as seguintes restrições (**1,5 Val**):
a) o primeiro é um número primo de um algarismo, ou seja, menor que 10;
b) o segundo é um número inteiro entre 0 e 100;
c) o terceiro é um número impar positivo menor que 10;
d) o segundo algarismo do segundo número é igual ao valor do primeiro número;
e) o primeiro número é maior que o terceiro;
f) os três números são diferentes;
g) a soma dos três números é um quadrado perfeito.
NOTA: para cada restrição que coloque, assinale com um comentário a(s) alínea(s) a que se refere.

7) Considere o seguinte predicado (**1,5 Val**):

```
p2(Vars, LO) :-
      Vars = [A,B,C,D,E,F,G,H],
      domain(Vars,1,10000),
      sum(Vars, #=, Sum),
      labeling(LO, Vars).
```

a) Se o objectivo for maximizar o somatório de `Vars`, qual deverá ser o valor de `LO` de modo a fazê-lo e de forma eficiente? Justifique.

b) Suponha que imediatamente antes da instrução de `labeling` eram colocadas as seguintes restrições:

```
D #= C - E,
D #< A,
D #> H,
```

Alteraria o valor de `LO` definido na alínea a)? Justifique e caso alterasse indique o novo valor de `LO` justificando-o também.

8) Imagine que quer colocar um conjunto de 6 peças em linha, de forma consecutiva. As peças podem estar juntas ou espaçadas, só não podem estar sobrepostas. As peças têm *1cm* de altura e a sua largura é a seguinte:

1-(*2cm*); 2-(*3cm*); 3-(*5cm*); 4-(*1cm*); 5-(*3cm*); 6-(*6cm*).

Implemente um predicado que indique como colocar as peças obedecendo às seguintes restrições (**1,5 Val**):

a) As peças devem caber num espaço com *25cm* de largura e *1cm* de altura.
b) A distância entre as peças 1 e 6 tem de ser superior a *5cm*.
c) A peça 5 tem de ser colocada numa posição mais avançada que a da peça 1.
d) Se a peça 3 aparecer depois da 1 deve ser colocada a *4cm* da origem.

NOTA: para cada restrição que coloque, assinale com um comentário a(s) alínea(s) a que se refere.

9) Uma máquina pode fabricar 4 tipos de peças. No entanto, existem restrições em relação à sequência de peças que pode ser fabricada. Adicionalmente, o custo de fabricar uma peça depende da peça que foi fabricada anteriormente. A tabela seguinte sumariza os custos de fabrico (exemplo: fabricar a peça 1 depois da 2 custa 3€, e não é possível fabricar as peças 2 e 3 imediatamente depois de se ter fabricado a peça 2):

		Peça Procedente			
		1	**2**	**3**	**4**
Peça Precedente	**1**	-	5	5	7
	2	3	-	-	4
	3	2	-	-	6
	4	9	5	-	-

Assuma que a primeira peça a ser fabricada tem custo zero e que não há restrições em relação ao seu tipo. A máquina produz um número fixo de peças/dia, sendo que num dia tem que produzir:

- 5 a 15 peças do tipo 1;
- 2 a 6 peças do tipo 2;
- 5 a 10 peças do tipo 3;
- 7 a 12 peças do tipo 4.

a) Implemente o predicado `sequencia(?Seq,?Custo,+N)` que determine a sequência de peças a fabricar num dia (`Seq`) que minimize o custo de fabrico (`Custo`) sujeito às restrições acima mencionadas. O número de peças a produzir é dado pelo argumento `N` (**1,5 Val**). Exemplo:

```
|?- sequencia(Seq,Custo,20).
Seq = [4,1,3,1,3,1,3,4,1,3,4,1,3,4,2,4,2,4,2,4],
Custo = 101
```

b) A sequência 3-4-2 coloca a máquina sob stress de tal modo que a sequência 4-2 apenas pode ser executada no máximo uma vez depois da sequência 3-4-2 ter aparecido. Implemente esta restrição e indique por palavras onde é que a incluiria no predicado da alínea anterior (**1,0 Val**).