Prolog

Descrição: O exemplo clássico para determinar que se todo homem é mortal e se Sócrates é um homem, então Sócrates é mortal. Essas afirmações podem ser representadas através das fórmulas:

$$\forall x(homem(x) \rightarrow mortal(x))$$
$$homem(socrates)$$

a partir destas pode-se concluir:

$$mortal(socrates)$$

O programa Prolog que descreve estas relações pode ser representado como a seguir.
Código:

```
mortal(X) :-                    % Todos os homens são mortais
  homem(X),
  writef('%w%w%w', ['Sim, ', X, ' é mortal']).
homem(socrates).                % Sócrates é um homem
```

```
?- mortal(socrates).
Sim, socrates é mortal
Yes
```

Descrição: Dado um conjunto de animais determinar a cadeia alimentar de um animal qualquer.

O programa Prolog que descreve estas relações pode ser representado como a seguir.

```
animal(urso).
animal(peixe).
animal(peixinho).
animal(guaxinim).
animal(raposa).
animal(coelho).
animal(veado).
animal(lince).
planta(alga).
planta(grama).
```

```
come(urso, peixe).
come(peixe, peixinho).
come(peixinho, alga).
come(guaxinim, peixe).
come(urso, guaxinim).
come(urso, raposa).
come(raposa, coelho).
come(coelho, grama).
come(urso, veado).
come(veado, grama).
come(lince, veado).
```

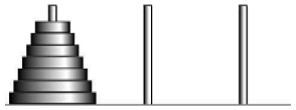
```
cadeia-alimentar(X, Z) :-
  come(X, Z).

cadeia-alimentar(X, Z) :-
  come(X, Y),
  cadeia-alimentar(Y, Z).
```

Consulta:

```
?- cadeia-alimentar(urso, Y).
Y = peixe ;
Y = guaxinim ;
Y = raposa ;
Y = veado ;
Y = peixinho ;
Y = alga ;
Y = peixe ;
Y = peixinho ;
Y = alga ;
Y = coelho ;
Y = grama ;
Y = grama ;
No
```

Descrição: Implementar um código para solucionar o jogo Torre de Hanoi, com n *peças. O jogo é formado por uma base contendo três pinos, em um destes pinos estão dispostos sete discos uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo, tal como ilustrado na Figura 5(a). O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma situação. O número de discos pode variar sendo que o mais simples contém apenas três. A Figura 5(a) ilustra um possível estado inicial e a Figura 5(b) ilustra um possível estado final para o estado inicial exibido.*



(a) Estado inicial para o jogo torre de hanoi.

(b) Estado final para o jogo torre de hanoi.

Código:

```
hanoi(N) :-
  move(N, esquerdo, central, direito).
move(0, _, _, _) :- !.
move(N, A, B, C) :-
  M is N-1,
  move(M, A, C, B),
  inform(A, B),
  move(M, C, B, A).
inform(X, Y) :-
  writef('%u%u%w%w', ['Mova o disco do pino ', X, ' para o pino ', Y]),
  nl.
```

Consulta:

```
?- hanoi(3).
Mova o disco do pino esquerdo para o pino central

Mova o disco do pino esquerdo para o pino direito
Mova o disco do pino central para o pino direito
Mova o disco do pino esquerdo para o pino central
Mova o disco do pino direito para o pino esquerdo
Mova o disco do pino direito para o pino central
Mova o disco do pino esquerdo para o pino central
Yes
```

Descrição: O bubble sort, *é o mais simples algoritmo de ordenação. O algoritmo consiste em percorrer um vetor várias vezes, em cada iteração o menor ou o maior elemento é colocado em sua posição correta no vetor.*

```
bubblesort(Lista_In, Lista_Out) :-
  append(L_Front, [A, B|Rest], Lista_In),  % Lista_In = [A + B|Rest]
  B < A,!,                                 % checa se B é menor que A
  append(L_Front, [B, A|Rest], L_Rest),    % L_Rest = [B + A|Rest]
  bubblesort(L_Rest, Lista_Out).           % utiliza e regra recursivamente
bubblesort(Lista, Lista).                  % Lista já está ordenada
```

```
?- bubblesort([2,3,1,7,5,4], L).
L = [1, 2, 3, 4, 5, 7] ;
No
```

Descrição: Dada uma lista de elementos, determine se ela é um palíndromo ou não. Um palíndromo é uma palavra ou número cuja leitura é a mesma, quer se faça da esquerda para a direita, quer se faça da direita para a esquerda;

```
palindrome([]).
palindrome([_]).
palindrome([F|R]) :-
    append(S,[F],R),
    palindrome(S).
```

```
?- palindrome([m,a,m]).
Yes
?- palindrome([m,a,a]).
No
?- palindrome([s,o,c,o,r,r,a,m,m,e,e,m,m,a,r,r,o,c,o,s]).
Yes
```

Descrição: Máximo Divisor Comum (M.D.C.). Dados dois inteiros positivos A e B, o eu máximo divisor comum, C, é o maior número que divide A e B sem deixar resto. Para encontrar o m.d.c de dois números é necessário trabalhar com três casos, são eles:

- *se A=B, então, C=A ou B;*
- *se A < B, então, C é igual ao maior divisor comum de A e B - A;*
- *se A > B, então, C é igual ao maior divisor comum de B e A - B.*

```
mdc(A, A, A). mdc(A, B, C) :-
  A < B,
  Temp is B - A,
  mdc(A, Temp, C).
mdc(A, B, C) :-
  A > B,
  Temp is A - B,
  mdc(Temp, B, C).
```

```
?- mdc(12, 18, C).
C = 6 ;
No
?- mdc(4,4,C).
C = 4 ;
No
```

Descrição: Cálculo de fatorial. Implementar um programa Prolog que realize o cálculo do fatorial de um número.

```
fatorial(0,1).
fatorial(N,F) :-
    N>0,
    N1 is N-1,
    factorial(N1,F1),
    F is N * F1.
```

```
?- fatorial(3,W).
W = 6 ;
No
?- fatorial(4, X).
X = 24 ;
No
```

Descrição: Implementar um programa que determina se um determinado dia faz parte de um dia da semana ou final de semana. Note que a determinação de uma categoria já exclui a possibilidade de que o elemento pertença a outra categoria, ou seja, não existe um dia que seja semana e final de semana.

```
semana(segunda).
semana(terca).
semana(quarta).
semana(quinta).
semana(sexta).
fimdesemana(sabado).
fimdesemana(domingo).

categoria(X) :-
  fimdesemana(X),
  writef('%w%w', [X, ' é um final de semana.']),
  !.
categoria(X) :-
  semana(X),
  writef('%w%w', [X, ' é um dia de semana.']),
  !.
```

```
?- categoria(segunda).
segunda é um dia de semana.
Yes.
?- categoria(domingo).
domingo é um final de semana.
Yes.
```

Exercícios

As questões 10.1 até 10.4 são relacionadas ao programa Prolog apresentado a seguir:

```
genitor( pam, bob).
genitor( tom, bob).
genitor( tom, liz).
genitor( bob, ann).
genitor( bob, pat).
genitor( pat, jim).
mulher(pam).
mulher(liz).
mulher(pat).
mulher(ann).
homem(tom).
homem(bob).
homem(jim).
prole(Y,X) :- genitor(X,Y).
mae(X,Y) :- genitor(X,Y), mulher(X).
avos(X,Z) :- genitor(X,Y), genitor(Y,Z).
irma(X,Y) :- genitor(Z,X), genitor(Z,Y), mulher(X), not(X = Y).
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).
```

Exercício 10.1. *Quais as respostas para as seguintes consultas?*

1. *?- genitor(X, jim).*
2. *?- genitor(jim, X).*
3. *?- genitor(pam, X), genitor(X, pat).*
4. *?- genitor(pam, X), genitor(X, Y), genitor(Y, jim).*

Exercício 10.2. *Formule consultas para descobrir:*

1. *Quem são os pais de Pat?*
2. *Liz possui filhos?*
3. *Quem é o avô de Pat?*
4. *Quem é a mãe de Jim?*
5. *Quem é o irmão de Bob?*
6. *Quem é a irmã de Pat?*

Exercício 10.3. *Formule regras para as seguintes relações:*

1. *tio(a);*
2. *irmão;*
3. *avós paternos;*
4. *avós maternos;*
5. *ascendente (o inverso de descendente);*
6. *primo(a), insira novos fatos para realizar consultas sobre esta relação.*

Exercício 10.4. *Em quais das seguintes consultas Prolog utiliza encadeamento para trás?*

1. *?- genitor(pam, bob).*
2. *?- mae(pam, bob).*
3. *?- avos(pam, ann).*
4. *?- avos(bob, jim).*

Exercício 10.5. *Considere as seguintes premissas:*

Todos os animais têm pele. Peixe é um tipo de animal, pássaros são outro tipo e mamíferos são um terceiro tipo. Normalmente, os peixes têm nadadeiras e podem nadar, enquanto os pássaros têm asas e podem voar. Se por um lado os pássaros e os peixes põem ovos, os mamíferos não põem. Embora tubarões sejam peixes, eles não põem ovos, seus filhotes nascem já formados. Salmão é um outro tipo de peixe, e é considerado uma delícia. O canário é um pássaro amarelo. Uma avestruz é um tipo de pássaro grande que não voa, apenas anda. Os mamíferos normalmente andam para se mover, como por exemplo uma vaca. As vacas dão leite, mas também servem elas mesmas de comida (carne). Contudo, nem todos os mamíferos andam para se mover. Por exemplo, o morcego voa.

Considere ainda que existem os seguinte animais:

1. *Piupiu, que é um canário.*
2. *Nemo, que é um peixe.*
3. *Tutu, que é um tubarão.*
4. *Mimosa, que é uma vaca.*
5. *Vamp, que é um morcego.*
6. *Xica, que é uma avestruz.*
7. *Aifred, que é um salmão.*

Defina fatos e regras Prolog que representam as premissas acima, e formule consultas Prolog para responder às seguintes perguntas:

1. *O piupiu voa?*
2. *Qual a cor do piupiu?*
3. *A Xica voa?*
4. *A Xica tem asas?*
5. *O Vamp voa? Tem asas? Poem ovos?*
6. *Quais os nomes dos animais que põem ovos?*
7. *Quais os nomes dos animais que são comestíveis?*
8. *Quais os nomes dos animais que se movem andando?*
9. *Quais os nomes dos animais que se movem nadando mas não põem ovos?*

As questões 10.6 até 10.8 são relacionadas ao programa Prolog apresentado a seguir:

```
animal(urso).
animal(peixe).
animal(peixinho).
animal(guaxinim).
animal(raposa).
animal(coelho).
animal(veado).
animal(lince).
planta(alga).
planta(grama).

come(urso, peixe).
come(peixe, peixinho).
come(peixinho, alga).
come(guaxinim, peixe).
come(urso, guaxinim).
come(urso, raposa).
come(raposa, coelho).
come(coelho, grama).
come(urso, veado).
come(veado, grama).
come(lince, veado).

presa(X) :- come(_, X), animal(X).
```

Exercício 10.6. *Quais as respostas para as seguintes consultas?*

1. *?- come(urso, peixinho).*
2. *?- come(raposa, coelho).*
3. *?- come(guaxinim, X).*
4. *?- come(X, grama).*
5. *?- come(urso, X), come(X, coelho).*
6. *?- presa(X), not(come(raposa, X)).*

Exercício 10.7. *Formule regras para as seguintes relações:*

1. *herbívoro;*
2. *carnívoro.*

Exercício 10.8. *Utilizando a regra da questão anterior, elabore consultas para as seguintes perguntas:*

1. *Quais animais são herbívoros?*
2. *Quais animais são carnívoros?*
3. *Quais animais herbívoros são presas de uma raposa?*

Exercício 10.9. *Elabore um programa Prolog que forneça o nome da capital de qualquer estado brasileiro.*

Exercício 10.10. *Implemente um programa para determinar quais tipos sanguíneos podem doar/receber sangue de quais tipos. A tabela seguinte fornece a informação necessária para a implementação.*

Tabela 1: Tipos sanguíneos.

	A	B	AB	O
A	Doa/Recebe	-	Doa	Recebe
B	-	Doa/Recebe	Doa	Recebe
AB	Recebe	Recebe	Doa/Recebe	Recebe
O	Doa	Doa	Doa	Doa/Recebe

Exercício 10.11. *Defina:*

1. *um predicado que forneça a intersecção de duas listas;*
2. *um predicado que identifique se um conjunto de elementos está contido em uma lista;*
3. *dois predicados que identifiquem se uma lista possui tamanho par ou ímpar;*
4. *um predicado que escreva uma lista em ordem inversa. Dica: utilize concatenação;*
5. *um predicado que retorne o maior valor contido em uma lista numérica;*
6. *um predicado para obter a soma dos N primeiros números naturais.*

Exercício 10.12. *Implemente um programa que retorne a N-ésima potência de um número.*

Exercício 10.13. *As regras abaixo identificam se um número é positivo, negativo ou zero. Defina regras que realizem o mesmo procedimento, porém, de maneira mais eficiente. Dica: utilize cortes.*

```
checagem(N, positivo) :- N > 0.
checagem(0, zero).
checagem(N, negativo) :- N < 0.
```

Exercício 10.14. *Implemente um programa que classifique se uma pessoa é considerada: criança (idade <= 12), adolescente (12 < idade <= 18), adulto (18 < idade <= 65) ou idoso (65 < idade).*

Obs.: Utilize cortes para melhorar a eficiência do programa.

Exercício 10.15. *Implemente um programa que forneça o signo de uma pessoa, mediante a uma consulta do tipo* dataNascimento(DD,MM, Signo).

Obs.: Utilize cortes para melhorar a eficiência do programa.

Exercício 10.16. *Um estudante acostumado a usar linguagens procedimentais está desenvolvendo um compilador em Prolog. Uma das tarefas consiste em traduzir um código de erro para uma pseudo-descrição em português. O código por ele usado é:*

```
traduza(Codigo, Significado):- Codigo=1, Significado=integer_overflow.
traduza(Codigo, Significado):- Codigo=2, Significado=divisao_por_zero.
traduza(Codigo, Significado):- Codigo=3, Significado=id_desconhecido.
```

O código funciona, porém, pode ser implementado de outra maneira (mais descritiva e menos procedimental). Melhore o código.