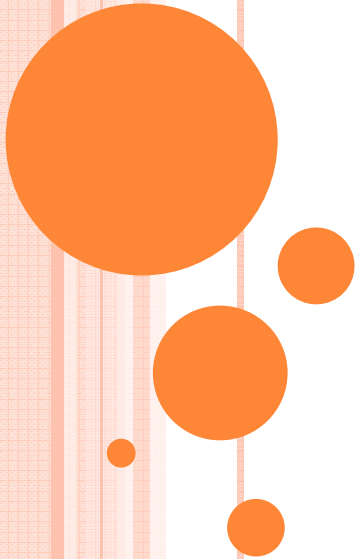
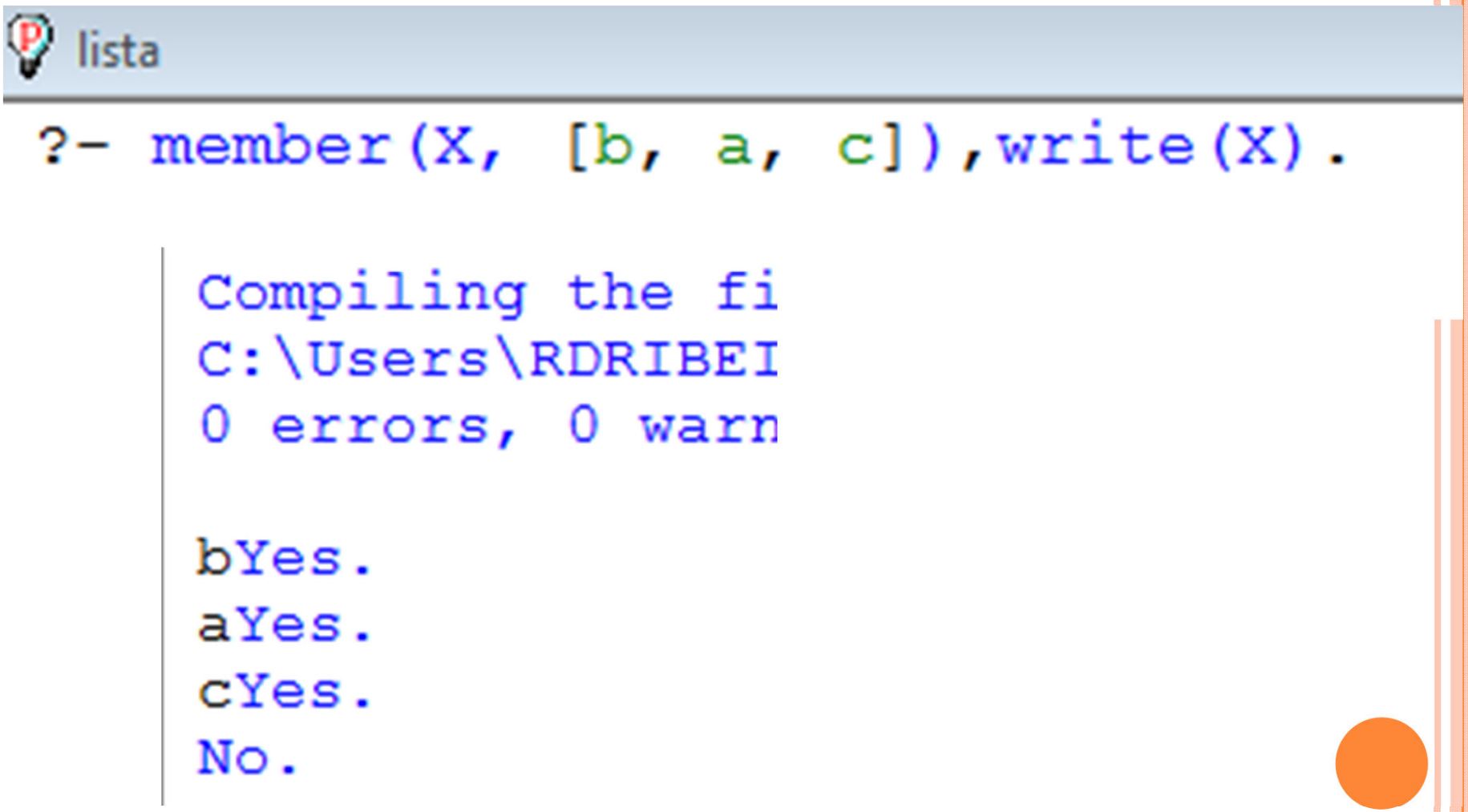


PROLOG

Rafael D. Ribeiro, M.Sc.
rafaeldiasribeiro@gmail.com
http://www.rafaeldiasribeiro.com.br

- Faça um programa que verifique se um valor X esta presente em uma lista L

Strawberry 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:

$$mortal(socrates)$$

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:

$$mortal(socrates)$$

```
homem(socrates).
mortal(X):-homem(X),write("SIM  "), write(X), write("  É MORTAL")

?- mortal(socrates).
```

```
Compiling the file:
C:\Users\RDRIBEIRO\Documents\EST
0 errors, 0 warnings.

SIM  socrates  É MORTALYes.
```

Criar as regras para operações em 2 números inteiros:
Adição – Subtração – Multiplicação – Divisão

Conta('operador', numero1, numero2)

O programa deve exibir a resposta.

Strawberry PROLOG

```
conta(X,Y,Z):- X = '+', R is Y+Z,write(R),!.
conta(X,Y,Z):-X = '-',R is Y-Z,write(R),!.
conta(X,Y,Z):-X = '*',R is Y*Z,write(R),!.
conta(X,Y,Z):- X = '/',R is Y//Z,write(R),!.

?-conta('+',2,3).
```

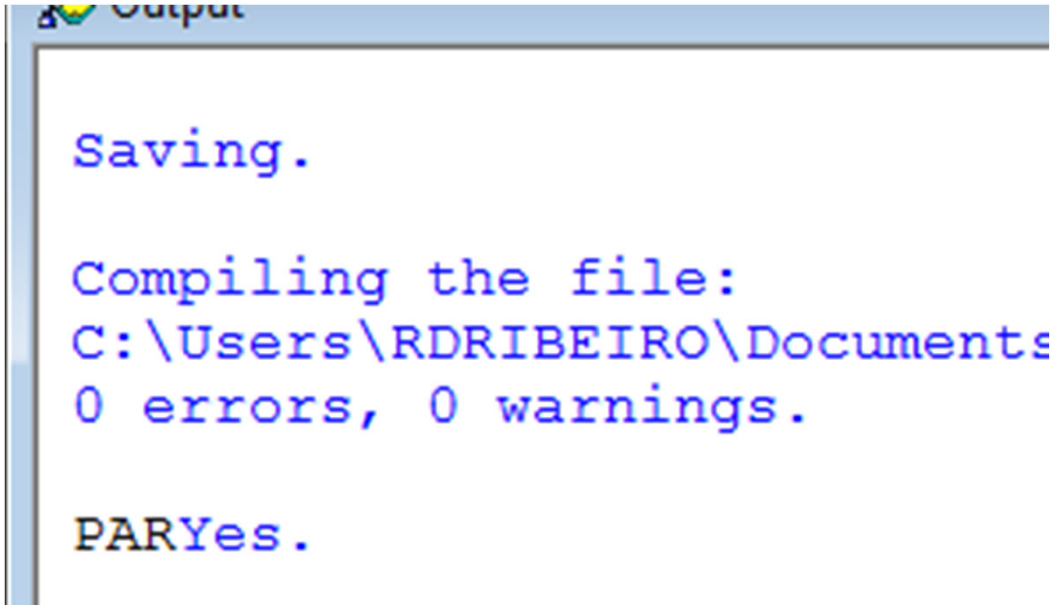
```
Compiling the file:
C:\Users\RDRIBEIRO\Docum
0 errors, 0 warnings.

6Yes.
```

Verificar se um número é PAR ou IMPAR e imprimir o resultado

Strawberry PROLOG

```
numero(X):- 0 is X mod 2, write("PAR"),!;write("IMPAR").
?-numero(2).
```



Criar as regras para verificar se um número é divisível por 2 e por 3 ou por 3 e por 5, se for escreva “DIVISIVEL” senão escreva “NÃO DIVISIVEL”

Strawberry PROLOG

```
numero(X):- 0 is X mod 2, 0 is X mod 3,write("DIVISIVEL");
0 is X mod 3, 0 is X mod 5,write("DIVISIVEL");
write("NAODIVISIVEL").

?-numero(30).
```

```
Compiling the file:
C:\Users\RDRIBEIRO\Documer
0 errors, 0 warnings.

DIVISIVELYes.
```

Em uma sequencia de 3 números inteiros determinar quem é o maior.

Strawberry PROLOG

```
maior(X,Y,Z):- X >= Y, X >=Z, write(X),!;Y>=X,Y>=Z,write(Y),!;Z>=X,Z>=Y ,write(Z),!.
?-maior(2,1,3).
```

```
Saving.

Compiling the file:
C:\Users\RDRIBEIRO\Documents\EST
0 errors, 0 warnings.

3Yes.
```

adiciona(X,L1,L2) – onde L2 é a lista que contém o elemento X e a lista L1.

Testar este predicado no interpretador Prolog, executando:

?- adiciona(1,[2,3],L).

?- adiciona(X,[2,3],[1,2,3]).

adiciona(X,L1,L2) – onde L2 é a lista que contém o elemento X e a lista L1.

Testar este predicado no interpretador Prolog, executando:

?- adiciona(1,[2,3],L).

?- adiciona(X,[2,3],[1,2,3]).

```
adiciona(X,L,[X|L]).


?- adiciona(1,[2,3],L).
?- adiciona(X,[2,3],[1,2,3]).
```

apaga(X,L1,L2) – onde L2 é a lista L1 sem o elemento X. Testar com:

?- apaga(a,[a,b,a,c],L).

?- apaga(a,L,[b,c]).

Strawberry PROLOG

apaga(X,L1,L2) – onde L2 é a lista L1 sem o elemento X. Testar com:
?- apaga(a,[a,b,a,c],L).
?- apaga(a,L,[b,c]).

```
apaga(X,[X|R],R).
apaga(X,[Y|R1],[Y|R2]):- apaga(X,R1,R2).

?- apaga(a,[a,b,a,c],L),write(L).
?- apaga(a,L,[b,c]),write(L).
```

membro(X,L) – que é verdadeiro se X pertencer à lista L. Testar com:

```
?- membro(b,[a,b,c]).
?- membro(X,[a,b,c]). % carregar em ;
?- findall(X,membro(X,[a,b,c]),L).
```

membro(X,L) – que é verdadeiro se X pertencer à lista L. Testar com:

?- membro(b,[a,b,c]).

?- membro(X,[a,b,c]). % carregar em ;

?- findall(X,membro(X,[a,b,c]),L).

```
membro( X, [X|_] ).
membro( X, [_|R] ) :- membro( X, R ).


?- membro(b,[a,b,c]).
?- membro(X,[a,b,c]). % carregar em ;
?- findall(X,membro(X,[a,b,c]),L).
```

concatena(L1,L2,L3) – onde L3 é resultado da junção das listas L2 e L1.

Testar com:

?- concatena([1,2],[3,4],L).

?- concatena([1,2],L,[1,2,3,4]).

?- concatena(L,[3,4],[1,2,3,4]).

Strawberry PROLOG

concatena(L1,L2,L3) – onde L3 é resultado da junção das listas L2 e L1.

Testar com:

?- concatena([1,2],[3,4],L).

?- concatena([1,2],L,[1,2,3,4]).

?- concatena(L,[3,4],[1,2,3,4]).

```
concatena([],L,L).
concatena([X|L1],L2,[X|L3]):- concatena(L1,L2,L3).


?- concatena([1,2],[3,4],L).
?- concatena([1,2],L,[1,2,3,4]).
?- concatena(L,[3,4],[1,2,3,4]).
```

comprimento(X,L) – onde X é o número de elementos da lista L.
Testar com:
?- comprimento(X,[a,b,c]).

Strawberry PROLOG

comprimento(X,L) – onde X é o número de elementos da lista L.
Testar com:
?- comprimento(X,[a,b,c]).

```
comprimento(0,[]).

comprimento(N,[_|R]):- comprimento(N1,R),N is 1 + N1.

?- comprimento(X,[a,b,c]),write(X).
```

media(X,L) – onde X é o valor médio da lista L (assumir que L contém
somente números). Testar com:
?- media(X,[1,2,3,4,5]).

Strawberry PROLOG

media(X,L) – onde X é o valor médio da lista L (assumir que L contém
somente números). Testar com:
?- media(X,[1,2,3,4,5]).

```
comprimento(0,[]).
comprimento(N,[_|R]):- comprimento(N1,R),N is 1 + N1.
somatorio(0,[]).
somatorio(X,[Y|R]):- somatorio(S,R),X is S+Y.
media(X,L):- comprimento(N,L),somatorio(S,L),X is S/N.

?- media(X,[1,2,3,4,5]),write(X).
```

Descrição: Calculo do Fatorial

Ex:

0 ! = 1
1 ! = 1*0!
2 ! = 2*1!
3 ! = 3*2!
4 ! = 4*3!

Strawberry PROLOG

```
fatorial(0,1).

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


?- fatorial(5,X),write(X).
```

Output

```
Compiling the fi
C:\Users\RDRIBEI
0 errors, 0 warn

120Yes.
```

Fatorial

Fatorial usando comando “is”:

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

Strawberry PROLOG

fatorial(3,F) – passo a passo (I)

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

---

fatorial(0,1). fail

fatorial (3,F) => N=3, N1 =2;

PILHA	
F	3 * F1

fatorial(3,F) – passo a passo (II)

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

---

fatorial(0,1). fail

fatorial (3,F) => N=3, N1 =1;

fatorial (2,F) => N=2, N1 =1;

PILHA	
F	2 * F1
F	3 * F1

fatorial(3,F) – passo a passo (III)

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

---

fatorial(0,1). fail

fatorial (3,F) => N=3, N1 =1;

fatorial (2,F) => N=2, N1 =1;

fatorial (1,F) => N=1, N1 =0;

PILHA	
F	1 * F1
F	2 * F1
F	3 * F1

fatorial(3,F) – passo a passo (IV)

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

---

fatorial(0,1). **true**

fatorial (3,F) => N=3, N1 =1;

fatorial (2,F) => N=2, N1 =1;

fatorial (1,F) => N=1, N1 =0;

fatorial(0,F) => F = 1

PILHA	
F	1
F	1 * F1
F	2 * F1
F	3 * F1

fatorial(3,F) – passo a passo (V)

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

---

fatorial (3,F) => N=3, N1 =1;

fatorial (2,F) => N=2, N1 =1;

fatorial (1,1) => N=1, N1 =0;

PILHA	
F = 1	1 * 1
F	2 * F1
F	3 * F1

fatorial(3,F) – passo a passo (VI)

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

---

fatorial (3,F) => N=3, N1 =1;

fatorial (2,2) => N=2, N1 =1;

PILHA	
F = 2	2 * 1
F	3 * F1

fatorial(3,F) – passo a passo (VII)

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

---

fatorial (3,6) => N=3, N1 =1;

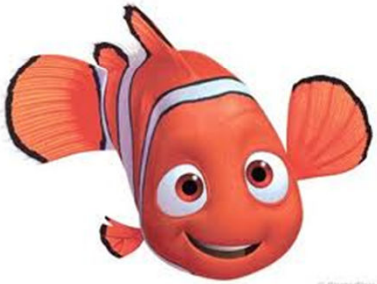
F = 6

PILHA	
F = 6	3 * 2

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

```
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).
```



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

```
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).
```

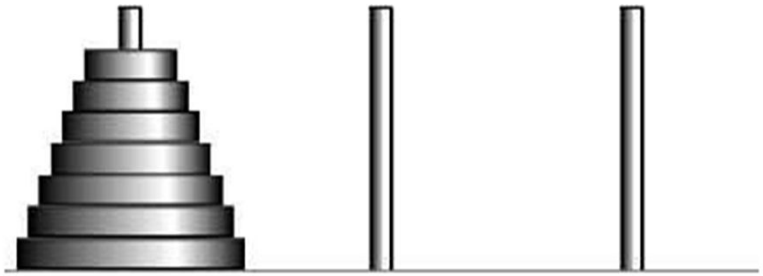
```
cadeia-alimentar(X, Z) :-
    come(X, Z).

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

Consulta:

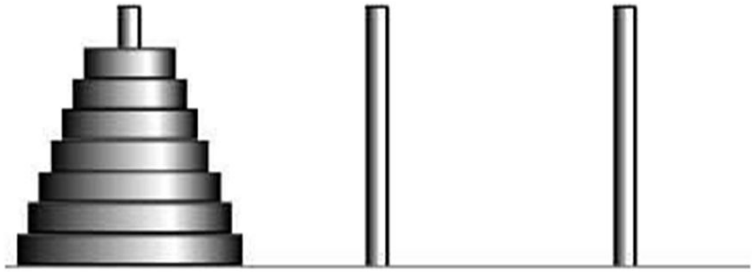
```
?- cadeia-alimentar(urso, Y).
Y = peixe ;
Y = guaxinim ;
Y = raposa ;
Y = veado ;
Y = peixinho ;
Y = alga ;
Y = peixe ;
Y = peixinho ;
```

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.



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

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

```
move(1,X,Y,_) :-
     write("Mova o disco do topo para "),
     write(X),
     write(" para "),
     write(Y),
     nl.
move(N,X,Y,Z) :-
     N>1,
     M is N-1,
     move(M,X,Z,Y),
     move(1,X,Y,_),
     move(M,Z,Y,X).

?- move(3,esquerda,direita,centro).
```

Output

```
Saving.

Compiling the file:
C:\Users\RDRIBEIRO\Documents\ESTACIO\Aulas\PROLOG\
0 errors, 0 warnings.

Mova o disco do topo para esquerda para direita
Mova o disco do topo para esquerda para centro
Mova o disco do topo para direita para centro
Mova o disco do topo para esquerda para direita
Mova o disco do topo para centro para esquerda
Mova o disco do topo para centro para direita
Mova o disco do topo para esquerda para direita
Yes.
```