Paradigmas de Programação

Prof. Maicon R. Zatelli

Prolog - Programação Lógica

Corte !

Universidade Federal de Santa Catarina

Florianópolis - Brasil

2018/1

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
max(X,Y,X) :- X >= Y.
max(X,Y,Y) :- X < Y.
```

Note que ambas as regras são mutualmente exclusivas, porém sem o uso do corte, ambas as regras serão testadas.

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ?
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ? creep
   Call: (7) 2>=1 ?
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ?
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Exit: (6) max(2, 1, 2) ?
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Exit: (6) max(2, 1, 2) ? creep
X = 2
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Exit: (6) max(2, 1, 2) ? creep
X = 2 ;
   Redo: (6) max(2, 1, _G338)
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Exit: (6) max(2, 1, 2) ? creep
X = 2 ;
   Redo: (6) max(2, 1, _G338) ? creep
   Call: (7) 2<1 ?
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Exit: (6) max(2, 1, 2) ? creep
X = 2 ;
   Redo: (6) max(2, 1, _G338) ? creep
   Call: (7) 2<1 ? creep
   Fail: (7) 2<1 ?
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Exit: (6) max(2, 1, 2) ? creep
X = 2 ;
   Redo: (6) max(2, 1, _G338) ? creep
   Call: (7) 2<1 ? creep
   Fail: (7) 2<1 ? creep
   Fail: (6) max(2, 1, _G338) ?
```

Prolog

O “corte” em Prolog tem por objetivo evitar *backtracking*.

```
[trace]  ?- max(2,1,X).
   Call: (6) max(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Exit: (6) max(2, 1, 2) ? creep
X = 2 ;
   Redo: (6) max(2, 1, _G338) ? creep
   Call: (7) 2<1 ? creep
   Fail: (7) 2<1 ? creep
   Fail: (6) max(2, 1, _G338) ? creep
false.
```

Prolog

Pode-se então introduzir o "corte", para evitar *backtracking* quando uma das regras já for satisfeita.

```
max2(X,Y,X) :- X >= Y, !.
max2(X,Y,Y).
```

Prolog

Pode-se então introduzir o "corte", para evitar *backtracking* quando uma das regras já for satisfeita.

```
[trace]  ?- max2(2,1,X).
```

Prolog

Pode-se então introduzir o "corte", para evitar *backtracking* quando uma das regras já for satisfeita.

```
[trace]  ?- max2(2,1,X).
   Call: (6) max2(2, 1, _G338) ?
```

Prolog

Pode-se então introduzir o "corte", para evitar *backtracking* quando uma das regras já for satisfeita.

```
[trace]  ?- max2(2,1,X).
   Call: (6) max2(2, 1, _G338) ? creep
   Call: (7) 2>=1 ?
```

Prolog

Pode-se então introduzir o "corte", para evitar *backtracking* quando uma das regras já for satisfeita.

```
[trace]  ?- max2(2,1,X).
   Call: (6) max2(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ?
```

Prolog

Pode-se então introduzir o "corte", para evitar *backtracking* quando uma das regras já for satisfeita.

```
[trace]  ?- max2(2,1,X).
   Call: (6) max2(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Exit: (6) max2(2, 1, 2) ?
```

Prolog

Pode-se então introduzir o "corte", para evitar *backtracking* quando uma das regras já for satisfeita.

```
[trace]  ?- max2(2,1,X).
   Call: (6) max2(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Exit: (6) max2(2, 1, 2) ? creep
X = 2.
```

Ou ainda, posso reescrever o mesmo da seguinte forma:

```
max3(X,Y,M) :- X >= Y, M = X, !;
               M = Y.
```

- Neste caso, se a primeira parte da minha regra for verdadeira, ou seja, se X for maior ou igual a Y e M pode ser unificado com X, então corto e a outra parte da minha regra (o OU) não será testada.
- Se X for menor que Y então a primeira parte da regra não será executada por completo, e então a segunda parte será testada.

Prolog

Ou ainda, posso reescrever o mesmo da seguinte forma:

```
[trace]  ?- max3(2,1,X).
```

Prolog

Ou ainda, posso reescrever o mesmo da seguinte forma:

```
[trace]  ?- max3(2,1,X).
   Call: (6) max3(2, 1, _G338) ?
```

Prolog

Ou ainda, posso reescrever o mesmo da seguinte forma:

```
[trace]  ?- max3(2,1,X).
   Call: (6) max3(2, 1, _G338) ? creep
   Call: (7) 2>=1 ?
```

Prolog

Ou ainda, posso reescrever o mesmo da seguinte forma:

```
[trace]  ?- max3(2,1,X).
   Call: (6) max3(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ?
```

Prolog

Ou ainda, posso reescrever o mesmo da seguinte forma:

```
[trace]  ?- max3(2,1,X).
   Call: (6) max3(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Call: (7) _G338=2 ?
```

Prolog

Ou ainda, posso reescrever o mesmo da seguinte forma:

```
[trace]  ?- max3(2,1,X).
   Call: (6) max3(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Call: (7) _G338=2 ? creep
   Exit: (7) 2=2 ?
```

Prolog

Ou ainda, posso reescrever o mesmo da seguinte forma:

```
[trace]  ?- max3(2,1,X).
   Call: (6) max3(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Call: (7) _G338=2 ? creep
   Exit: (7) 2=2 ? creep
   Exit: (6) max3(2, 1, 2) ?
```

Prolog

Ou ainda, posso reescrever o mesmo da seguinte forma:

```
[trace]  ?- max3(2,1,X).
   Call: (6) max3(2, 1, _G338) ? creep
   Call: (7) 2>=1 ? creep
   Exit: (7) 2>=1 ? creep
   Call: (7) _G338=2 ? creep
   Exit: (7) 2=2 ? creep
   Exit: (6) max3(2, 1, 2) ? creep
X = 2.
```

Prolog

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

- Se X ocorrer várias vezes, irá reportar todas as ocorrências de X na lista.

Prolog

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

- Se X ocorrer várias vezes, irá reportar todas as ocorrências de X na lista.

```
?- membro(1,[1,2,3,1]).
```

Prolog

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

- Se X ocorrer várias vezes, irá reportar todas as ocorrências de X na lista.

```
?- membro(1,[1,2,3,1]).
true
```

Prolog

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

- Se X ocorrer várias vezes, irá reportar todas as ocorrências de X na lista.

```
?- membro(1,[1,2,3,1]).
true ;
true
```

Prolog

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

- Se X ocorrer várias vezes, irá reportar todas as ocorrências de X na lista.

```
?- membro(1,[1,2,3,1]).
true ;
true ;
false.
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (9) membro(1, [1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (9) membro(1, [1]) ? creep
   Call: (10) membro(1, []) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (9) membro(1, [1]) ? creep
   Call: (10) membro(1, []) ? creep
   Fail: (10) membro(1, []) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (9) membro(1, [1]) ? creep
   Call: (10) membro(1, []) ? creep
   Fail: (10) membro(1, []) ? creep
   Fail: (9) membro(1, [1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (9) membro(1, [1]) ? creep
   Call: (10) membro(1, []) ? creep
   Fail: (10) membro(1, []) ? creep
   Fail: (9) membro(1, [1]) ? creep
   Fail: (8) membro(1, [3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (9) membro(1, [1]) ? creep
   Call: (10) membro(1, []) ? creep
   Fail: (10) membro(1, []) ? creep
   Fail: (9) membro(1, [1]) ? creep
   Fail: (8) membro(1, [3, 1]) ? creep
   Fail: (7) membro(1, [2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (9) membro(1, [1]) ? creep
   Call: (10) membro(1, []) ? creep
   Fail: (10) membro(1, []) ? creep
   Fail: (9) membro(1, [1]) ? creep
   Fail: (8) membro(1, [3, 1]) ? creep
   Fail: (7) membro(1, [2, 3, 1]) ? creep
   Fail: (6) membro(1, [1, 2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro(1,[1,2,3,1]).
   Call: (6) membro(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (6) membro(1, [1, 2, 3, 1]) ? creep
   Call: (7) membro(1, [2, 3, 1]) ? creep
   Call: (8) membro(1, [3, 1]) ? creep
   Call: (9) membro(1, [1]) ? creep
   Exit: (9) membro(1, [1]) ? creep
   Exit: (8) membro(1, [3, 1]) ? creep
   Exit: (7) membro(1, [2, 3, 1]) ? creep
   Exit: (6) membro(1, [1, 2, 3, 1]) ? creep
true ;
   Redo: (9) membro(1, [1]) ? creep
   Call: (10) membro(1, []) ? creep
   Fail: (10) membro(1, []) ? creep
   Fail: (9) membro(1, [1]) ? creep
   Fail: (8) membro(1, [3, 1]) ? creep
   Fail: (7) membro(1, [2, 3, 1]) ? creep
   Fail: (6) membro(1, [1, 2, 3, 1]) ? creep
false.
```

```
membro2(X,[X|_]) :- !.
membro2(X,[_|T]) :- membro2(X,T).
```

- Se X for encontrado, não faz mais backtracking, ou seja, não continua mais procurando outras ocorrências de X na lista.

Prolog

```
membro2(X,[X|_]) :- !.
membro2(X,[_|T]) :- membro2(X,T).
```

- Se X for encontrado, não faz mais backtracking, ou seja, não continua mais procurando outras ocorrências de X na lista.

```
?- membro2(1,[1,2,3,1]).
```

Prolog

```
membro2(X,[X|_]) :- !.
membro2(X,[_|T]) :- membro2(X,T).
```

- Se X for encontrado, não faz mais backtracking, ou seja, não continua mais procurando outras ocorrências de X na lista.

```
?- membro2(1,[1,2,3,1]).
true.
```

Prolog

```
[trace]  ?- membro2(1,[1,2,3,1]).
```

Prolog

```
[trace]  ?- membro2(1,[1,2,3,1]).
   Call: (6) membro2(1, [1, 2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro2(1,[1,2,3,1]).
   Call: (6) membro2(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro2(1, [1, 2, 3, 1]) ?
```

Prolog

```
[trace]  ?- membro2(1,[1,2,3,1]).
   Call: (6) membro2(1, [1, 2, 3, 1]) ? creep
   Exit: (6) membro2(1, [1, 2, 3, 1]) ? creep
true.
```

Prolog - Alguns Links Úteis

- http://www.swi-prolog.org/pldoc/man?predicate=!/0
- http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse44