Paradigmas de Programação

Prof. Maicon R. Zatelli

Prolog - Programação Lógica
Listas

Universidade Federal de Santa Catarina
Florianópolis - Brasil
2018/1

Prolog

Lista vazia

- [ ]
- [1, 2, 3, 4] - uma lista de números
- ['bob', 'bia', 'bin'] - uma lista de nomes de pessoas
- [bob, bia, bin] - uma lista de átomos
- [ponto(1,2), ponto(4,1), ponto(6,3)] - uma lista de pontos

Cabeça, cauda e outros padrões para manipular listas

- [H | T] - ao menos um item na lista
- [H1, H2 | T] - ao menos dois itens na lista
- [H1, H2] - exatamante dois itens na lista
- [H] - exatamente um item na lista

Prolog

```
comprimento([],0).
comprimento([H|T],C) :- comprimento(T,CT), C is CT + 1.
```

- O comprimento de uma lista é 0 se ela é vazia, caso contrário, o comprimento dela é 1 mais o comprimento da cauda.

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

Prolog

```
comprimento([],0).
comprimento([H|T],C) :- comprimento(T,CT), C is CT + 1.
```

- O comprimento de uma lista é 0 se ela é vazia, caso contrário, o comprimento dela é 1 mais o comprimento da cauda.

```
?- comprimento([1,2,3,4,5],X).
X = 5.
```

Prolog

```
comprimento([],0).
comprimento([H|T],C) :- comprimento(T,CT), C is CT + 1.
```

- O comprimento de uma lista é 0 se ela é vazia, caso contrário, o comprimento dela é 1 mais o comprimento da cauda.

```
?- comprimento([1,2,3,4,5],X).
X = 5.

?- comprimento([bob, bia, bin],3).
```

Prolog

```
comprimento([],0).
comprimento([H|T],C) :- comprimento(T,CT), C is CT + 1.
```

- O comprimento de uma lista é 0 se ela é vazia, caso contrário, o comprimento dela é 1 mais o comprimento da cauda.

```
?- comprimento([1,2,3,4,5],X).
X = 5.

?- comprimento([bob, bia, bin],3).
true.
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
true
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
true ;
true
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
true ;
true ;
false.
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
true ;
true ;
false.

?- membro(X,[bia,bia,bob,bin]).
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
true ;
true ;
false.

?- membro(X,[bia,bia,bob,bin]).
X = bia
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
true ;
true ;
false.

?- membro(X,[bia,bia,bob,bin]).
X = bia ;
X = bia
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
true ;
true ;
false.

?- membro(X,[bia,bia,bob,bin]).
X = bia ;
X = bia ;
X = bob
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
true ;
true ;
false.

?- membro(X,[bia,bia,bob,bin]).
X = bia ;
X = bia ;
X = bob ;
X = bin
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.

```
?- membro(bia,[bia, bob, bin]).
true ;
false.

?- membro(bia,[bia, bia, bob, bin]).
true ;
true ;
false.

?- membro(X,[bia,bia,bob,bin]).
X = bia ;
X = bia ;
X = bob ;
X = bin ;
false.
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.
- ! significa corte (*cut*) e serve para podar ramos na busca. Se já detectei que X é membro da lista, posso querer parar.

```
?- membro(bia,[bia, bob, bin]).
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.
- ! significa corte (*cut*) e serve para podar ramos na busca. Se já detectei que X é membro da lista, posso querer parar.

```
?- membro(bia,[bia, bob, bin]).
true.
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.
- ! significa corte (*cut*) e serve para podar ramos na busca. Se já detectei que X é membro da lista, posso querer parar.

```
?- membro(bia,[bia, bob, bin]).
true.

?- membro(bia,[bia, bia, bob, bin]).
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.
- ! significa corte (*cut*) e serve para podar ramos na busca. Se já detectei que X é membro da lista, posso querer parar.

```
?- membro(bia,[bia, bob, bin]).
true.

?- membro(bia,[bia, bia, bob, bin]).
true.
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.
- ! significa corte (*cut*) e serve para podar ramos na busca. Se já detectei que X é membro da lista, posso querer parar.

```
?- membro(bia,[bia, bob, bin]).
true.

?- membro(bia,[bia, bia, bob, bin]).
true.

?- membro(X,[bia,bia,bob,bin]).
```

Prolog

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

- Um elemento é membro de uma lista se ele é a cabeça da lista ou se ele é membro da cauda da lista.
- ! significa corte (*cut*) e serve para podar ramos na busca. Se já detectei que X é membro da lista, posso querer parar.

```
?- membro(bia,[bia, bob, bin]).
true.

?- membro(bia,[bia, bia, bob, bin]).
true.

?- membro(X,[bia,bia,bob,bin]).
X = bia.
```

Prolog

Como comparar se duas listas são iguais?

Prolog

Como comparar se duas listas são iguais?

```
igual(L,L).
```

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

Prolog

Como comparar se duas listas são iguais?

```
igual(L,L).
```

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

Prolog

Como comparar se duas listas são iguais?

```
igual(L,L).
```

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

?- igual([a,2,3],[a,2,3]).
```

Prolog

Como comparar se duas listas são iguais?

```
igual(L,L).
```

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

?- igual([a,2,3],[a,2,3]).
true.
```

Como comparar se duas listas são iguais?

```
igual(L,L).
```

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

?- igual([a,2,3],[a,2,3]).
true.

?- igual([a,2,ponto(1,2)],[a,2,ponto(1,2)]).
```

Prolog

Como comparar se duas listas são iguais?

```
igual(L,L).
```

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

?- igual([a,2,3],[a,2,3]).
true.

?- igual([a,2,ponto(1,2)],[a,2,ponto(1,2)]).
true.
```

Prolog

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

- Se insiro X em uma lista, a lista resultante contém X na cabeça.

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

Prolog

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

- Se insiro X em uma lista, a lista resultante contém X na cabeça.

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

Prolog

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

- Se insiro X em uma lista, a lista resultante contém X na cabeça.

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

?- inserirElemento([4,5,6],[2,3,4],L).
```

Prolog

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

- Se insiro X em uma lista, a lista resultante contém X na cabeça.

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

?- inserirElemento([4,5,6],[2,3,4],L).
L = [[4, 5, 6], 2, 3, 4].
```

Prolog

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

- Se insiro X em uma lista, a lista resultante contém X na cabeça.

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

?- inserirElemento([4,5,6],[2,3,4],L).
L = [[4, 5, 6], 2, 3, 4].

?- inserirElemento(2,[],L).
```

Prolog

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

- Se insiro X em uma lista, a lista resultante contém X na cabeça.

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

?- inserirElemento([4,5,6],[2,3,4],L).
L = [[4, 5, 6], 2, 3, 4].

?- inserirElemento(2,[],L).
L = [2].
```

Prolog

```
ultimo([X], X).
ultimo([H,H2|T], X) :- ultimo([H2|T],X).
```

- Se a lista é formada por um único elemento, então ele é o último elemento.
- Se a lista possui mais de um elemento, o último elemento é o último da cauda.

```
?- ultimo([1,2,3,4],X).
```

Prolog

```
ultimo([X], X).
ultimo([H,H2|T], X) :- ultimo([H2|T],X).
```

- Se a lista é formada por um único elemento, então ele é o último elemento.
- Se a lista possui mais de um elemento, o último elemento é o último da cauda.

```
?- ultimo([1,2,3,4],X).
X = 4 .
```

Prolog

```
ultimo([X], X).
ultimo([H,H2|T], X) :- ultimo([H2|T],X).
```

- Se a lista é formada por um único elemento, então ele é o último elemento.
- Se a lista possui mais de um elemento, o último elemento é o último da cauda.

```
?- ultimo([1,2,3,4],X).
X = 4 .

?- ultimo([],X).
```

Prolog

```
ultimo([X], X).
ultimo([H,H2|T], X) :- ultimo([H2|T],X).
```

- Se a lista é formada por um único elemento, então ele é o último elemento.
- Se a lista possui mais de um elemento, o último elemento é o último da cauda.

```
?- ultimo([1,2,3,4],X).
X = 4 .

?- ultimo([],X).
false.
```

Prolog

```
dobro([],[]).
dobro([H1|T1],[H2|T2]) :- H2 is H1 * 2, dobro(T1,T2).
```

- O dobro de uma lista vazia é uma lista vazia. Caso contrário, o dobro dela é o dobro da cabeça e o dobro da cauda.

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

Prolog

```
dobro([],[]).
dobro([H1|T1],[H2|T2]) :- H2 is H1 * 2, dobro(T1,T2).
```

- O dobro de uma lista vazia é uma lista vazia. Caso contrário, o dobro dela é o dobro da cabeça e o dobro da cauda.

```
?- dobro([1,2,3],X).
X = [2, 4, 6].
```

Prolog

```
tamanhoPar([]) :- !.
tamanhoPar([_|T]) :- tamanhoImpar(T).

tamanhoImpar([_]) :- !.
tamanhoImpar([_|T]) :- tamanhoPar(T).
```

- Uma lista tem tamanho par se é vazia ou se a cauda tem tamanho ímpar.
- Uma lista tem tamanho ímpar se possui um único elemento ou se a cauda tem tamanho par.

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

Prolog

```
tamanhoPar([]) :- !.
tamanhoPar([_|T]) :- tamanhoImpar(T).

tamanhoImpar([_]) :- !.
tamanhoImpar([_|T]) :- tamanhoPar(T).
```

- Uma lista tem tamanho par se é vazia ou se a cauda tem tamanho ímpar.
- Uma lista tem tamanho ímpar se possui um único elemento ou se a cauda tem tamanho par.

```
?- tamanhoPar([1,2,3,4]).
true.
```

Prolog

```
tamanhoPar([]) :- !.
tamanhoPar([_|T]) :- tamanhoImpar(T).

tamanhoImpar([_]) :- !.
tamanhoImpar([_|T]) :- tamanhoPar(T).
```

- Uma lista tem tamanho par se é vazia ou se a cauda tem tamanho ímpar.
- Uma lista tem tamanho ímpar se possui um único elemento ou se a cauda tem tamanho par.

```
?- tamanhoPar([1,2,3,4]).
true.

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

Prolog

```
tamanhoPar([]) :- !.
tamanhoPar([_|T]) :- tamanhoImpar(T).

tamanhoImpar([_]) :- !.
tamanhoImpar([_|T]) :- tamanhoPar(T).
```

- Uma lista tem tamanho par se é vazia ou se a cauda tem tamanho ímpar.
- Uma lista tem tamanho ímpar se possui um único elemento ou se a cauda tem tamanho par.

```
?- tamanhoPar([1,2,3,4]).
true.

?- tamanhoImpar([1,2,3,4]).
false.
```

Prolog

```
tamanhoPar([]) :- !.
tamanhoPar([_|T]) :- tamanhoImpar(T).

tamanhoImpar([_]) :- !.
tamanhoImpar([_|T]) :- tamanhoPar(T).
```

- Uma lista tem tamanho par se é vazia ou se a cauda tem tamanho ímpar.
- Uma lista tem tamanho ímpar se possui um único elemento ou se a cauda tem tamanho par.

```
?- tamanhoPar([1,2,3,4]).
true.

?- tamanhoImpar([1,2,3,4]).
false.

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

Prolog

```
tamanhoPar([]) :- !.
tamanhoPar([_|T]) :- tamanhoImpar(T).

tamanhoImpar([_]) :- !.
tamanhoImpar([_|T]) :- tamanhoPar(T).
```

- Uma lista tem tamanho par se é vazia ou se a cauda tem tamanho ímpar.
- Uma lista tem tamanho ímpar se possui um único elemento ou se a cauda tem tamanho par.

```
?- tamanhoPar([1,2,3,4]).
true.

?- tamanhoImpar([1,2,3,4]).
false.

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

Prolog

```
enesimo(0,[H|_],H).
enesimo(I,[_|T],X) :- I2 is I - 1, enesimo(I2,T,X).
```

- O 0-ésimo item da lista é a cabeça da lista, se ela tiver um ou mais itens.
- O I-ésimo item da lista é o (I-1)-ésimo item da cauda da lista.

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

Prolog

```
enesimo(0,[H|_],H).
enesimo(I,[_|T],X) :- I2 is I - 1, enesimo(I2,T,X).
```

- O 0-ésimo item da lista é a cabeça da lista, se ela tiver um ou mais itens.
- O I-ésimo item da lista é o (I-1)-ésimo item da cauda da lista.

```
?- enesimo(2,[1,2,3,4,5],X).
X = 3 .
```

Prolog

```
enesimo(0,[H|_],H).
enesimo(I,[_|T],X) :- I2 is I - 1, enesimo(I2,T,X).
```

- O 0-ésimo item da lista é a cabeça da lista, se ela tiver um ou mais itens.
- O I-ésimo item da lista é o (I-1)-ésimo item da cauda da lista.

```
?- enesimo(2,[1,2,3,4,5],X).
X = 3 .

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

Prolog

```
enesimo(0,[H|_],H).
enesimo(I,[_|T],X) :- I2 is I - 1, enesimo(I2,T,X).
```

- O 0-ésimo item da lista é a cabeça da lista, se ela tiver um ou mais itens.
- O I-ésimo item da lista é o (I-1)-ésimo item da cauda da lista.

```
?- enesimo(2,[1,2,3,4,5],X).
X = 3 .

?- enesimo(2,[1,2],X).
false.
```

Prolog

```
enesimo(0,[H|_],H).
enesimo(I,[_|T],X) :- I2 is I - 1, enesimo(I2,T,X).
```

- O 0-ésimo item da lista é a cabeça da lista, se ela tiver um ou mais itens.
- O I-ésimo item da lista é o (I-1)-ésimo item da cauda da lista.

```
?- enesimo(2,[1,2,3,4,5],X).
X = 3 .

?- enesimo(2,[1,2],X).
false.

?- enesimo(0,[1],X).
```

Prolog

```
enesimo(0,[H|_],H).
enesimo(I,[_|T],X) :- I2 is I - 1, enesimo(I2,T,X).
```

- O 0-ésimo item da lista é a cabeça da lista, se ela tiver um ou mais itens.
- O I-ésimo item da lista é o (I-1)-ésimo item da cauda da lista.

```
?- enesimo(2,[1,2,3,4,5],X).
X = 3 .

?- enesimo(2,[1,2],X).
false.

?- enesimo(0,[1],X).
X = 1 .
```

Prolog

```
enesimo(0,[H|_],H).
enesimo(I,[_|T],X) :- I2 is I - 1, enesimo(I2,T,X).
```

- O 0-ésimo item da lista é a cabeça da lista, se ela tiver um ou mais itens.
- O I-ésimo item da lista é o (I-1)-ésimo item da cauda da lista.

```
?- enesimo(2,[1,2,3,4,5],X).
X = 3 .

?- enesimo(2,[1,2],X).
false.

?- enesimo(0,[1],X).
X = 1 .

?- enesimo(0,[],X).
```

Prolog

```
enesimo(0,[H|_],H).
enesimo(I,[_|T],X) :- I2 is I - 1, enesimo(I2,T,X).
```

- O 0-ésimo item da lista é a cabeça da lista, se ela tiver um ou mais itens.
- O I-ésimo item da lista é o (I-1)-ésimo item da cauda da lista.

```
?- enesimo(2,[1,2,3,4,5],X).
X = 3 .

?- enesimo(2,[1,2],X).
false.

?- enesimo(0,[1],X).
X = 1 .

?- enesimo(0,[],X).
false.
```

Prolog

```
remover(X,[],[]).
remover(X,[X|T],T2) :- remover(X, T, T2).
remover(X,[H|T],[H|T2]) :- X \== H, remover(X, T, T2).
```

- Se tento remover um item X de uma lista vazia, retorno a lista vazia.
- Se X é a cabeça da minha lista, ignoro ele, e retorno a lista resultante da remoção de X da cauda da lista.
- Se X não é a cabeça da minha lista, retorno ele e a lista resultante da remoção de X da cauda da lista.

```
?- remover(3,[1,2,3,4,5,3],L).
```

Prolog

```
remover(X,[],[]).
remover(X,[X|T],T2) :- remover(X, T, T2).
remover(X,[H|T],[H|T2]) :- X \== H, remover(X, T, T2).
```

- Se tento remover um item X de uma lista vazia, retorno a lista vazia.
- Se X é a cabeça da minha lista, ignoro ele, e retorno a lista resultante da remoção de X da cauda da lista.
- Se X não é a cabeça da minha lista, retorno ele e a lista resultante da remoção de X da cauda da lista.

```
?- remover(3,[1,2,3,4,5,3],L).
L = [1, 2, 4, 5] .
```

Prolog

```
remover(X,[],[]).
remover(X,[X|T],T2) :- remover(X, T, T2).
remover(X,[H|T],[H|T2]) :- X \== H, remover(X, T, T2).
```

- Se tento remover um item X de uma lista vazia, retorno a lista vazia.
- Se X é a cabeça da minha lista, ignoro ele, e retorno a lista resultante da remoção de X da cauda da lista.
- Se X não é a cabeça da minha lista, retorno ele e a lista resultante da remoção de X da cauda da lista.

```
?- remover(3,[1,2,3,4,5,3],L).
L = [1, 2, 4, 5] .

?- remover(3,[3],L).
```

Prolog

```
remover(X,[],[]).
remover(X,[X|T],T2) :- remover(X, T, T2).
remover(X,[H|T],[H|T2]) :- X \== H, remover(X, T, T2).
```

- Se tento remover um item X de uma lista vazia, retorno a lista vazia.
- Se X é a cabeça da minha lista, ignoro ele, e retorno a lista resultante da remoção de X da cauda da lista.
- Se X não é a cabeça da minha lista, retorno ele e a lista resultante da remoção de X da cauda da lista.

```
?- remover(3,[1,2,3,4,5,3],L).
L = [1, 2, 4, 5] .

?- remover(3,[3],L).
L = [] .
```

Prolog

```
remover(X,[],[]).
remover(X,[X|T],T2) :- remover(X, T, T2).
remover(X,[H|T],[H|T2]) :- X \== H, remover(X, T, T2).
```

- Se tento remover um item X de uma lista vazia, retorno a lista vazia.
- Se X é a cabeça da minha lista, ignoro ele, e retorno a lista resultante da remoção de X da cauda da lista.
- Se X não é a cabeça da minha lista, retorno ele e a lista resultante da remoção de X da cauda da lista.

```
?- remover(3,[1,2,3,4,5,3],L).
L = [1, 2, 4, 5] .

?- remover(3,[3],L).
L = [] .

?- remover(3,[],L).
```

Prolog

```
remover(X,[],[]).
remover(X,[X|T],T2) :- remover(X, T, T2).
remover(X,[H|T],[H|T2]) :- X \== H, remover(X, T, T2).
```

- Se tento remover um item X de uma lista vazia, retorno a lista vazia.
- Se X é a cabeça da minha lista, ignoro ele, e retorno a lista resultante da remoção de X da cauda da lista.
- Se X não é a cabeça da minha lista, retorno ele e a lista resultante da remoção de X da cauda da lista.

```
?- remover(3,[1,2,3,4,5,3],L).
L = [1, 2, 4, 5] .

?- remover(3,[3],L).
L = [] .

?- remover(3,[],L).
L = [] .
```

Prolog

```
concatena([],L,L).
concatena([H|T],L2,[H|LContatenada]) :- concatena(T,L2,LContatenada).
```

- Se a primeira lista é vazia, a concatenação dela com outra é a outra lista.
- Se a primeira lista não é vazia, então adiciono a cabeça dela na lista resultante e concateno a cauda da primeira lista com a segunda lista.

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

Prolog

```
concatena([],L,L).
concatena([H|T],L2,[H|LContatenada]) :- concatena(T,L2,LContatenada).
```

- Se a primeira lista é vazia, a concatenação dela com outra é a outra lista.
- Se a primeira lista não é vazia, então adiciono a cabeça dela na lista resultante e concateno a cauda da primeira lista com a segunda lista.

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

Prolog

```
concatena([],L,L).
concatena([H|T],L2,[H|LContatenada]) :- concatena(T,L2,LContatenada).
```

- Se a primeira lista é vazia, a concatenação dela com outra é a outra lista.
- Se a primeira lista não é vazia, então adiciono a cabeça dela na lista resultante e concateno a cauda da primeira lista com a segunda lista.

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

?- concatena([],[2,3,4,5],L).
```

Prolog

```
concatena([],L,L).
concatena([H|T],L2,[H|LContatenada]) :- concatena(T,L2,LContatenada).
```

- Se a primeira lista é vazia, a concatenação dela com outra é a outra lista.
- Se a primeira lista não é vazia, então adiciono a cabeça dela na lista resultante e concateno a cauda da primeira lista com a segunda lista.

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

?- concatena([],[2,3,4,5],L).
L = [2, 3, 4, 5].
```

Prolog

```
concatena([],L,L).
concatena([H|T],L2,[H|LContatenada]) :- concatena(T,L2,LContatenada).
```

- Se a primeira lista é vazia, a concatenação dela com outra é a outra lista.
- Se a primeira lista não é vazia, então adiciono a cabeça dela na lista resultante e concateno a cauda da primeira lista com a segunda lista.

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

?- concatena([],[2,3,4,5],L).
L = [2, 3, 4, 5].

?- concatena([1,2,3,4],[5,6,7],L).
```

Prolog

```
concatena([],L,L).
concatena([H|T],L2,[H|LContatenada]) :- concatena(T,L2,LContatenada).
```

- Se a primeira lista é vazia, a concatenação dela com outra é a outra lista.
- Se a primeira lista não é vazia, então adiciono a cabeça dela na lista resultante e concateno a cauda da primeira lista com a segunda lista.

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

?- concatena([],[2,3,4,5],L).
L = [2, 3, 4, 5].

?- concatena([1,2,3,4],[5,6,7],L).
L = [1, 2, 3, 4, 5, 6, 7].
```

Lista para Conjunto - deseja-se remover os elementos duplicados.

```
listaParaConjunto([],[]).
listaParaConjunto([H|T],[H|L]):- not(membro(H,T)), listaParaConjunto(T,L).
listaParaConjunto([H|T],L):- membro(H,T), listaParaConjunto(T,L).
```

- Se a lista é vazia, então não há elementos duplicados.
- Se a lista não é vazia e a cabeça da lista não aparece na cauda, então ela é membro do conjunto.
- Se a lista não é vazia e a cabeça aparece na cauda, então ignoramos a cabeça e convertemos a cauda para conjunto.

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

Lista para Conjunto - deseja-se remover os elementos duplicados.

```
listaParaConjunto([],[]).
listaParaConjunto([H|T],[H|L]):- not(membro(H,T)), listaParaConjunto(T,L).
listaParaConjunto([H|T],L):- membro(H,T), listaParaConjunto(T,L).
```

- Se a lista é vazia, então não há elementos duplicados.
- Se a lista não é vazia e a cabeça da lista não aparece na cauda, então ela é membro do conjunto.
- Se a lista não é vazia e a cabeça aparece na cauda, então ignoramos a cabeça e convertemos a cauda para conjunto.

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

Lista para Conjunto - deseja-se remover os elementos duplicados.

```
listaParaConjunto([],[]).
listaParaConjunto([H|T],[H|L]):- not(membro(H,T)), listaParaConjunto(T,L).
listaParaConjunto([H|T],L):- membro(H,T), listaParaConjunto(T,L).
```

- Se a lista é vazia, então não há elementos duplicados.
- Se a lista não é vazia e a cabeça da lista não aparece na cauda, então ela é membro do conjunto.
- Se a lista não é vazia e a cabeça aparece na cauda, então ignoramos a cabeça e convertemos a cauda para conjunto.

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

?- listaParaConjunto([1],L).
```

Lista para Conjunto - deseja-se remover os elementos duplicados.

```
listaParaConjunto([],[]).
listaParaConjunto([H|T],[H|L]):- not(membro(H,T)), listaParaConjunto(T,L).
listaParaConjunto([H|T],L):- membro(H,T), listaParaConjunto(T,L).
```

- Se a lista é vazia, então não há elementos duplicados.
- Se a lista não é vazia e a cabeça da lista não aparece na cauda, então ela é membro do conjunto.
- Se a lista não é vazia e a cabeça aparece na cauda, então ignoramos a cabeça e convertemos a cauda para conjunto.

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

?- listaParaConjunto([1],L).
L = [1] .
```

Lista para Conjunto - deseja-se remover os elementos duplicados.

```
listaParaConjunto([],[]).
listaParaConjunto([H|T],[H|L]):- not(membro(H,T)), listaParaConjunto(T,L).
listaParaConjunto([H|T],L):- membro(H,T), listaParaConjunto(T,L).
```

- Se a lista é vazia, então não há elementos duplicados.
- Se a lista não é vazia e a cabeça da lista não aparece na cauda, então ela é membro do conjunto.
- Se a lista não é vazia e a cabeça aparece na cauda, então ignoramos a cabeça e convertemos a cauda para conjunto.

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

?- listaParaConjunto([1],L).
L = [1] .

?- listaParaConjunto([],L).
```

Lista para Conjunto - deseja-se remover os elementos duplicados.

```
listaParaConjunto([],[]).
listaParaConjunto([H|T],[H|L]):- not(membro(H,T)), listaParaConjunto(T,L).
listaParaConjunto([H|T],L):- membro(H,T), listaParaConjunto(T,L).
```

- Se a lista é vazia, então não há elementos duplicados.
- Se a lista não é vazia e a cabeça da lista não aparece na cauda, então ela é membro do conjunto.
- Se a lista não é vazia e a cabeça aparece na cauda, então ignoramos a cabeça e convertemos a cauda para conjunto.

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

?- listaParaConjunto([1],L).
L = [1] .

?- listaParaConjunto([],L).
L = [].
```

Subconjuntos

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

Subconjuntos

```
?- subconjunto([1,2,3],L).
L = [1, 2, 3]
```

Subconjuntos

```
?- subconjunto([1,2,3],L).
L = [1, 2, 3] ;
L = [1, 2]
```

Subconjuntos

```
?- subconjunto([1,2,3],L).
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3]
```

Subconjuntos

```
?- subconjunto([1,2,3],L).
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1]
```

Subconjuntos

```
?- subconjunto([1,2,3],L).
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1] ;
L = [2, 3]
```

Subconjuntos

```
?- subconjunto([1,2,3],L).
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1] ;
L = [2, 3] ;
L = [2]
```

Subconjuntos

```
?- subconjunto([1,2,3],L).
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1] ;
L = [2, 3] ;
L = [2] ;
L = [3]
```

Subconjuntos

```
?- subconjunto([1,2,3],L).
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1] ;
L = [2, 3] ;
L = [2] ;
L = [3] ;
L = [].
```

Subconjuntos

```
?- subconjunto([1,2,3],L).
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1] ;
L = [2, 3] ;
L = [2] ;
L = [3] ;
L = [].
```

- Cada item aparece e não aparece com todos os subconjuntos dos demais itens.

Subconjuntos

```
subconjunto([],[]). %Subconjunto de um conjunto vazio é ele próprio
subconjunto([H|S],[H|Sub]) :- subconjunto(S,Sub).
subconjunto([H|S],Sub) :- subconjunto(S,Sub).
```

- Os subconjuntos de um conjunto são formados pela cabeça do conjunto e pelos subconjuntos da cauda, ou apenas pelos subconjuntos da cauda, ignorando a cabeça.

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

Subconjuntos

```
subconjunto([],[]). %Subconjunto de um conjunto vazio é ele próprio
subconjunto([H|S],[H|Sub]) :- subconjunto(S,Sub).
subconjunto([H|S],Sub) :- subconjunto(S,Sub).
```

- Os subconjuntos de um conjunto são formados pela cabeça do conjunto e pelos subconjuntos da cauda, ou apenas pelos subconjuntos da cauda, ignorando a cabeça.

```
?- subconjunto([1,2,3],L).
%a cabeça faz parte dos subconjuntos
L = [1, 2, 3]
```

Subconjuntos

```
subconjunto([],[]). %Subconjunto de um conjunto vazio é ele próprio
subconjunto([H|S],[H|Sub]) :- subconjunto(S,Sub).
subconjunto([H|S],Sub) :- subconjunto(S,Sub).
```

- Os subconjuntos de um conjunto são formados pela cabeça do conjunto e pelos subconjuntos da cauda, ou apenas pelos subconjuntos da cauda, ignorando a cabeça.

```
?- subconjunto([1,2,3],L).
%a cabeça faz parte dos subconjuntos
L = [1, 2, 3] ;
L = [1, 2]
```

Subconjuntos

```
subconjunto([],[]). %Subconjunto de um conjunto vazio é ele próprio
subconjunto([H|S],[H|Sub]) :- subconjunto(S,Sub).
subconjunto([H|S],Sub) :- subconjunto(S,Sub).
```

- Os subconjuntos de um conjunto são formados pela cabeça do conjunto e pelos subconjuntos da cauda, ou apenas pelos subconjuntos da cauda, ignorando a cabeça.

```
?- subconjunto([1,2,3],L).
%a cabeça faz parte dos subconjuntos
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3]
```

Subconjuntos

```
subconjunto([],[]). %Subconjunto de um conjunto vazio é ele próprio
subconjunto([H|S],[H|Sub]) :- subconjunto(S,Sub).
subconjunto([H|S],Sub) :- subconjunto(S,Sub).
```

- Os subconjuntos de um conjunto são formados pela cabeça do conjunto e pelos subconjuntos da cauda, ou apenas pelos subconjuntos da cauda, ignorando a cabeça.

```
?- subconjunto([1,2,3],L).
%a cabeça faz parte dos subconjuntos
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1]
```

Subconjuntos

```
subconjunto([],[]). %Subconjunto de um conjunto vazio é ele próprio
subconjunto([H|S],[H|Sub]) :- subconjunto(S,Sub).
subconjunto([H|S],Sub) :- subconjunto(S,Sub).
```

- Os subconjuntos de um conjunto são formados pela cabeça do conjunto e pelos subconjuntos da cauda, ou apenas pelos subconjuntos da cauda, ignorando a cabeça.

```
?- subconjunto([1,2,3],L).
%a cabeça faz parte dos subconjuntos
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1] ;
%a cabeça não faz parte dos subconjuntos
L = [2, 3]
```

Subconjuntos

```
subconjunto([],[]). %Subconjunto de um conjunto vazio é ele próprio
subconjunto([H|S],[H|Sub]) :- subconjunto(S,Sub).
subconjunto([H|S],Sub) :- subconjunto(S,Sub).
```

- Os subconjuntos de um conjunto são formados pela cabeça do conjunto e pelos subconjuntos da cauda, ou apenas pelos subconjuntos da cauda, ignorando a cabeça.

```
?- subconjunto([1,2,3],L).
%a cabeça faz parte dos subconjuntos
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1] ;
%a cabeça não faz parte dos subconjuntos
L = [2, 3] ;
L = [2]
```

Subconjuntos

```
subconjunto([],[]). %Subconjunto de um conjunto vazio é ele próprio
subconjunto([H|S],[H|Sub]) :- subconjunto(S,Sub).
subconjunto([H|S],Sub) :- subconjunto(S,Sub).
```

- Os subconjuntos de um conjunto são formados pela cabeça do conjunto e pelos subconjuntos da cauda, ou apenas pelos subconjuntos da cauda, ignorando a cabeça.

```
?- subconjunto([1,2,3],L).
%a cabeça faz parte dos subconjuntos
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1] ;
%a cabeça não faz parte dos subconjuntos
L = [2, 3] ;
L = [2] ;
L = [3]
```

Subconjuntos

```
subconjunto([],[]). %Subconjunto de um conjunto vazio é ele próprio
subconjunto([H|S],[H|Sub]) :- subconjunto(S,Sub).
subconjunto([H|S],Sub) :- subconjunto(S,Sub).
```

- Os subconjuntos de um conjunto são formados pela cabeça do conjunto e pelos subconjuntos da cauda, ou apenas pelos subconjuntos da cauda, ignorando a cabeça.

```
?- subconjunto([1,2,3],L).
%a cabeça faz parte dos subconjuntos
L = [1, 2, 3] ;
L = [1, 2] ;
L = [1, 3] ;
L = [1] ;
%a cabeça não faz parte dos subconjuntos
L = [2, 3] ;
L = [2] ;
L = [3] ;
L = [].
```

Intersecção

```
interseccao([],_,[]).
interseccao([H|T],S,[H|R]) :- membro(H,S), intersseccao(T,S,R).
interseccao([H|T],S,R) :- not(membro(H,S)), intersseccao(T,S,R).
```

- A intersecção de um conjunto vazio com qualquer conjunto é sempre um conjunto vazio.
- A cabeça do primeiro conjunto pertence à intersecção se ela pertence ao outro conjunto.
- Caso contrário, ela não pertence à intersecção.

```
?- interseccao([1,2,3,4],[2,3,5,6,7,8],L).
```

Intersecção

```
interseccao([],_,[]).
interseccao([H|T],S,[H|R]) :- membro(H,S), intersseccao(T,S,R).
interseccao([H|T],S,R) :- not(membro(H,S)), intersseccao(T,S,R).
```

- A intersecção de um conjunto vazio com qualquer conjunto é sempre um conjunto vazio.
- A cabeça do primeiro conjunto pertence à intersecção se ela pertence ao outro conjunto.
- Caso contrário, ela não pertence à intersecção.

```
?- interseccao([1,2,3,4],[2,3,5,6,7,8],L).
L = [2, 3] .
```

Intersecção

```
interseccao([],_,[]).
interseccao([H|T],S,[H|R]) :- membro(H,S), intersseccao(T,S,R).
interseccao([H|T],S,R) :- not(membro(H,S)), intersseccao(T,S,R).
```

- A intersecção de um conjunto vazio com qualquer conjunto é sempre um conjunto vazio.
- A cabeça do primeiro conjunto pertence à intersecção se ela pertence ao outro conjunto.
- Caso contrário, ela não pertence à intersecção.

```
?- interseccao([1,2,3,4],[2,3,5,6,7,8],L).
L = [2, 3] .

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

Intersecção

```
interseccao([],_,[]).
interseccao([H|T],S,[H|R]) :- membro(H,S), intersseccao(T,S,R).
interseccao([H|T],S,R) :- not(membro(H,S)), intersseccao(T,S,R).
```

- A intersecção de um conjunto vazio com qualquer conjunto é sempre um conjunto vazio.
- A cabeça do primeiro conjunto pertence à intersecção se ela pertence ao outro conjunto.
- Caso contrário, ela não pertence à intersecção.

```
?- interseccao([1,2,3,4],[2,3,5,6,7,8],L).
L = [2, 3] .

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

Intersecção

```
interseccao([],_,[]).
interseccao([H|T],S,[H|R]) :- membro(H,S), intersseccao(T,S,R).
interseccao([H|T],S,R) :- not(membro(H,S)), intersseccao(T,S,R).
```

- A intersecção de um conjunto vazio com qualquer conjunto é sempre um conjunto vazio.
- A cabeça do primeiro conjunto pertence à intersecção se ela pertence ao outro conjunto.
- Caso contrário, ela não pertence à intersecção.

```
?- interseccao([1,2,3,4],[2,3,5,6,7,8],L).
L = [2, 3] .

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

?- interseccao([],[2,3,5,6,7,8],L).
```

Intersecção

```
interseccao([],_,[]).
interseccao([H|T],S,[H|R]) :- membro(H,S), intersseccao(T,S,R).
interseccao([H|T],S,R) :- not(membro(H,S)), intersseccao(T,S,R).
```

- A intersecção de um conjunto vazio com qualquer conjunto é sempre um conjunto vazio.
- A cabeça do primeiro conjunto pertence à intersecção se ela pertence ao outro conjunto.
- Caso contrário, ela não pertence à intersecção.

```
?- interseccao([1,2,3,4],[2,3,5,6,7,8],L).
L = [2, 3] .

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

?- interseccao([],[2,3,5,6,7,8],L).
L = [].
```

Permutação

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

Permutação

```
?- permutacao([1,2,3],L).
L = [1, 2, 3]
```

Permutação

```
?- permutacao([1,2,3],L).
L = [1, 2, 3] ;
L = [2, 1, 3]
```

Permutação

```
?- permutacao([1,2,3],L).
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1]
```

Permutação

```
?- permutacao([1,2,3],L).
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
L = [1, 3, 2]
```

Permutação

```
?- permutacao([1,2,3],L).
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
L = [1, 3, 2] ;
L = [3, 1, 2]
```

Permutação

```
?- permutacao([1,2,3],L).
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
L = [1, 3, 2] ;
L = [3, 1, 2] ;
L = [3, 2, 1]
```

Permutação

```
?- permutacao([1,2,3],L).
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
L = [1, 3, 2] ;
L = [3, 1, 2] ;
L = [3, 2, 1] ;
false.
```

Permutação

```
?- permutacao([1,2,3],L).
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
L = [1, 3, 2] ;
L = [3, 1, 2] ;
L = [3, 2, 1] ;
false.
```

- Cada item aparece em todas as posições da lista.
- Cada lista onde o item é inserido é uma permutação dos demais itens.

Permutação

Assuma a permutação de [1,2,3]. Primeiro fazemos a permutação de [2,3] que resulta em [2,3] e [3,2], depois inserimos 1 em todas as posições de [2,3] e [3,2].

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

Permutação

Assuma a permutação de [1,2,3]. Primeiro fazemos a permutação de [2,3] que resulta em [2,3] e [3,2], depois inserimos 1 em todas as posições de [2,3] e [3,2].

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3]
```

Permutação

Assuma a permutação de [1,2,3]. Primeiro fazemos a permutação de [2,3] que resulta em [2,3] e [3,2], depois inserimos 1 em todas as posições de [2,3] e [3,2].

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3]
```

Permutação

Assuma a permutação de [1,2,3]. Primeiro fazemos a permutação de [2,3] que resulta em [2,3] e [3,2], depois inserimos 1 em todas as posições de [2,3] e [3,2].

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1]
```

Permutação

Assuma a permutação de [1,2,3]. Primeiro fazemos a permutação de [2,3] que resulta em [2,3] e [3,2], depois inserimos 1 em todas as posições de [2,3] e [3,2].

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
%inserindo 1 em [3,2]
L = [1, 3, 2]
```

Permutação

Assuma a permutação de [1,2,3]. Primeiro fazemos a permutação de [2,3] que resulta em [2,3] e [3,2], depois inserimos 1 em todas as posições de [2,3] e [3,2].

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
%inserindo 1 em [3,2]
L = [1, 3, 2] ;
L = [3, 1, 2]
```

Permutação

Assuma a permutação de [1,2,3]. Primeiro fazemos a permutação de [2,3] que resulta em [2,3] e [3,2], depois inserimos 1 em todas as posições de [2,3] e [3,2].

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
%inserindo 1 em [3,2]
L = [1, 3, 2] ;
L = [3, 1, 2] ;
L = [3, 2, 1]
```

- Cada item aparece em todas as posições da lista.
- Cada lista onde o item é inserido é uma permutação dos demais itens.

Permutação

Assuma a permutação de [1,2,3]. Primeiro fazemos a permutação de [2,3] que resulta em [2,3] e [3,2], depois inserimos 1 em todas as posições de [2,3] e [3,2].

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
%inserindo 1 em [3,2]
L = [1, 3, 2] ;
L = [3, 1, 2] ;
L = [3, 2, 1] ;
false.
```

- Cada item aparece em todas as posições da lista.
- Cada lista onde o item é inserido é uma permutação dos demais itens.

Permutação

```
permutacao([],[]).
permutacao([H|T],L) :- permutacao(T,L1),
                       inserir(H,L1,L).
```

- A permutação de uma lista vazia é uma lista vazia.
- A permutação de uma lista não vazia é a permutação da cauda da lista, com a subsequente inserção de X nela em todas as posições.

Permutação

```
permutacao([],[]).
permutacao([H|T],L) :- permutacao(T,L1),
                       inserir(H,L1,L).
```

- A permutação de uma lista vazia é uma lista vazia.
- A permutação de uma lista não vazia é a permutação da cauda da lista, com a subsequente inserção de X nela em todas as posições.

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

Permutação

```
permutacao([],[]).
permutacao([H|T],L) :- permutacao(T,L1),
                       inserir(H,L1,L).
```

- A permutação de uma lista vazia é uma lista vazia.
- A permutação de uma lista não vazia é a permutação da cauda da lista, com a subsequente inserção de X nela em todas as posições.

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3]
```

Permutação

```
permutacao([],[]).
permutacao([H|T],L) :- permutacao(T,L1),
                       inserir(H,L1,L).
```

- A permutação de uma lista vazia é uma lista vazia.
- A permutação de uma lista não vazia é a permutação da cauda da lista, com a subsequente inserção de X nela em todas as posições.

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3]
```

Permutação

```
permutacao([],[]).
permutacao([H|T],L) :- permutacao(T,L1),
                       inserir(H,L1,L).
```

- A permutação de uma lista vazia é uma lista vazia.
- A permutação de uma lista não vazia é a permutação da cauda da lista, com a subsequente inserção de X nela em todas as posições.

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1]
```

Permutação

```
permutacao([],[]).
permutacao([H|T],L) :- permutacao(T,L1),
                       inserir(H,L1,L).
```

- A permutação de uma lista vazia é uma lista vazia.
- A permutação de uma lista não vazia é a permutação da cauda da lista, com a subsequente inserção de X nela em todas as posições.

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
%inserindo 1 em [3,2]
L = [1, 3, 2]
```

Permutação

```
permutacao([],[]).
permutacao([H|T],L) :- permutacao(T,L1),
                       inserir(H,L1,L).
```

- A permutação de uma lista vazia é uma lista vazia.
- A permutação de uma lista não vazia é a permutação da cauda da lista, com a subsequente inserção de X nela em todas as posições.

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
%inserindo 1 em [3,2]
L = [1, 3, 2] ;
L = [3, 1, 2]
```

Permutação

```
permutacao([],[]).
permutacao([H|T],L) :- permutacao(T,L1),
                       inserir(H,L1,L).
```

- A permutação de uma lista vazia é uma lista vazia.
- A permutação de uma lista não vazia é a permutação da cauda da lista, com a subsequente inserção de X nela em todas as posições.

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
%inserindo 1 em [3,2]
L = [1, 3, 2] ;
L = [3, 1, 2] ;
L = [3, 2, 1]
```

Permutação

```
permutacao([],[]).
permutacao([H|T],L) :- permutacao(T,L1),
                       inserir(H,L1,L).
```

- A permutação de uma lista vazia é uma lista vazia.
- A permutação de uma lista não vazia é a permutação da cauda da lista, com a subsequente inserção de X nela em todas as posições.

```
?- permutacao([1,2,3],L).
%inserindo 1 em [2,3]
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [2, 3, 1] ;
%inserindo 1 em [3,2]
L = [1, 3, 2] ;
L = [3, 1, 2] ;
L = [3, 2, 1] ;
false.
```

Permutação

```
inserir(X,[],[X]).
inserir(X,[H|T],[X,H|T]).
inserir(X,[H|T],[H|L]) :- inserir(X,T,L).
```

- Se insiro X numa lista vazia, a lista conterá apenas X.
- Se a lista não é vazia, posso inserir X antes da cabeça da lista.
- Também posso inserir X na cauda da lista, mantendo a cabeça da lista e inserindo X na cauda.

```
?- inserir(12,[1,2,3],L).
```

Permutação

```
inserir(X,[],[X]).
inserir(X,[H|T],[X,H|T]).
inserir(X,[H|T],[H|L]) :- inserir(X,T,L).
```

- Se insiro X numa lista vazia, a lista conterá apenas X.
- Se a lista não é vazia, posso inserir X antes da cabeça da lista.
- Também posso inserir X na cauda da lista, mantendo a cabeça da lista e inserindo X na cauda.

```
?- inserir(12,[1,2,3],L).
L = [12, 1, 2, 3]
```

Permutação

```
inserir(X,[],[X]).
inserir(X,[H|T],[X,H|T]).
inserir(X,[H|T],[H|L]) :- inserir(X,T,L).
```

- Se insiro X numa lista vazia, a lista conterá apenas X.
- Se a lista não é vazia, posso inserir X antes da cabeça da lista.
- Também posso inserir X na cauda da lista, mantendo a cabeça da lista e inserindo X na cauda.

```
?- inserir(12,[1,2,3],L).
L = [12, 1, 2, 3] ;
L = [1, 12, 2, 3]
```

Permutação

```
inserir(X,[],[X]).
inserir(X,[H|T],[X,H|T]).
inserir(X,[H|T],[H|L]) :- inserir(X,T,L).
```

- Se insiro X numa lista vazia, a lista conterá apenas X.
- Se a lista não é vazia, posso inserir X antes da cabeça da lista.
- Também posso inserir X na cauda da lista, mantendo a cabeça da lista e inserindo X na cauda.

```
?- inserir(12,[1,2,3],L).
L = [12, 1, 2, 3] ;
L = [1, 12, 2, 3] ;
L = [1, 2, 12, 3]
```

Permutação

```
inserir(X,[],[X]).
inserir(X,[H|T],[X,H|T]).
inserir(X,[H|T],[H|L]) :- inserir(X,T,L).
```

- Se insiro X numa lista vazia, a lista conterá apenas X.
- Se a lista não é vazia, posso inserir X antes da cabeça da lista.
- Também posso inserir X na cauda da lista, mantendo a cabeça da lista e inserindo X na cauda.

```
?- inserir(12,[1,2,3],L).
L = [12, 1, 2, 3] ;
L = [1, 12, 2, 3] ;
L = [1, 2, 12, 3] ;
L = [1, 2, 3, 12]
```

Permutação

```
inserir(X,[],[X]).
inserir(X,[H|T],[X,H|T]).
inserir(X,[H|T],[H|L]) :- inserir(X,T,L).
```

- Se insiro X numa lista vazia, a lista conterá apenas X.
- Se a lista não é vazia, posso inserir X antes da cabeça da lista.
- Também posso inserir X na cauda da lista, mantendo a cabeça da lista e inserindo X na cauda.

```
?- inserir(12,[1,2,3],L).
L = [12, 1, 2, 3] ;
L = [1, 12, 2, 3] ;
L = [1, 2, 12, 3] ;
L = [1, 2, 3, 12] ;
false.
```

Permutação

```
inserir(X,[],[X]).
inserir(X,[H|T],[X,H|T]).
inserir(X,[H|T],[H|L]) :- inserir(X,T,L).
```

- Se insiro X numa lista vazia, a lista conterá apenas X.
- Se a lista não é vazia, posso inserir X antes da cabeça da lista.
- Também posso inserir X na cauda da lista, mantendo a cabeça da lista e inserindo X na cauda.

```
?- inserir(12,[1,2,3],L).
L = [12, 1, 2, 3] ;
L = [1, 12, 2, 3] ;
L = [1, 2, 12, 3] ;
L = [1, 2, 3, 12] ;
false.

?- inserir(12,[],L).
```

Permutação

```
inserir(X,[],[X]).
inserir(X,[H|T],[X,H|T]).
inserir(X,[H|T],[H|L]) :- inserir(X,T,L).
```

- Se insiro X numa lista vazia, a lista conterá apenas X.
- Se a lista não é vazia, posso inserir X antes da cabeça da lista.
- Também posso inserir X na cauda da lista, mantendo a cabeça da lista e inserindo X na cauda.

```
?- inserir(12,[1,2,3],L).
L = [12, 1, 2, 3] ;
L = [1, 12, 2, 3] ;
L = [1, 2, 12, 3] ;
L = [1, 2, 3, 12] ;
false.

?- inserir(12,[],L).
L = [12]
```

Permutação

```
inserir(X,[],[X]).
inserir(X,[H|T],[X,H|T]).
inserir(X,[H|T],[H|L]) :- inserir(X,T,L).
```

- Se insiro X numa lista vazia, a lista conterá apenas X.
- Se a lista não é vazia, posso inserir X antes da cabeça da lista.
- Também posso inserir X na cauda da lista, mantendo a cabeça da lista e inserindo X na cauda.

```
?- inserir(12,[1,2,3],L).
L = [12, 1, 2, 3] ;
L = [1, 12, 2, 3] ;
L = [1, 2, 12, 3] ;
L = [1, 2, 3, 12] ;
false.

?- inserir(12,[],L).
L = [12] ;
false.
```

Ordenação

```
ordenada([]).
ordenada([_]).
ordenada([H1,H2|T]) :- H1 =< H2, ordenada([H2|T]).
```

- Se uma lista é vazia, ela está ordenada.
- Se uma lista contém um único elemento, ela está ordenada.
- Se uma lista contém dois ou mais elementos, ela estará ordenada se o primeiro elemento for menor ou igual ao segundo e o restante da lista a partir do segundo elemento está também ordenado.

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

Ordenação

```
ordenada([]).
ordenada([_]).
ordenada([H1,H2|T]) :- H1 =< H2, ordenada([H2|T]).
```

- Se uma lista é vazia, ela está ordenada.
- Se uma lista contém um único elemento, ela está ordenada.
- Se uma lista contém dois ou mais elementos, ela estará ordenada se o primeiro elemento for menor ou igual ao segundo e o restante da lista a partir do segundo elemento está também ordenado.

```
?- ordenada([1,2,3,4]).
true .
```

Ordenação

```
ordenada([]).
ordenada([_]).
ordenada([H1,H2|T]) :- H1 =< H2, ordenada([H2|T]).
```

- Se uma lista é vazia, ela está ordenada.
- Se uma lista contém um único elemento, ela está ordenada.
- Se uma lista contém dois ou mais elementos, ela estará ordenada se o primeiro elemento for menor ou igual ao segundo e o restante da lista a partir do segundo elemento está também ordenado.

```
?- ordenada([1,2,3,4]).
true .

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

Ordenação

```
ordenada([]).
ordenada([_]).
ordenada([H1,H2|T]) :- H1 =< H2, ordenada([H2|T]).
```

- Se uma lista é vazia, ela está ordenada.
- Se uma lista contém um único elemento, ela está ordenada.
- Se uma lista contém dois ou mais elementos, ela estará ordenada se o primeiro elemento for menor ou igual ao segundo e o restante da lista a partir do segundo elemento está também ordenado.

```
?- ordenada([1,2,3,4]).
true .

?- ordenada([3,4,1,2]).
false.
```

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

- Se uma lista é vazia, ela está ordenada.
- Caso contrário, se ela contém um ou mais elementos, ordeno a cauda e adiciono a cabeça na cauda já ordenada.

Continua ...

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

- Se adiciono um elemento X em uma lista vazia, formo uma lista com um único elemento.
- Se adiciono um elemento X em uma lista com ao menos um elemento e X é maior que a cabeça da lista, então adiciono X na cauda da lista.
- Se adiciono um elemento X em uma lista com ao menos um elemento e X é menor ou igual a cabeça da lista, então X deve ser inserido antes da cabeça da lista.

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
```

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
L = [1, 2, 3, 3, 4, 5] .
```

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
L = [1, 2, 3, 3, 4, 5] .

?- add(0, [1,2,4,5], L).
```

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
L = [1, 2, 3, 3, 4, 5] .

?- add(0, [1,2,4,5], L).
L = [0, 1, 2, 4, 5].
```

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
L = [1, 2, 3, 3, 4, 5] .

?- add(0, [1,2,4,5], L).
L = [0, 1, 2, 4, 5].

?- add(6, [1,2,4,5], L).
```

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
L = [1, 2, 3, 3, 4, 5] .

?- add(0, [1,2,4,5], L).
L = [0, 1, 2, 4, 5].

?- add(6, [1,2,4,5], L).
L = [1, 2, 4, 5, 6] .
```

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
L = [1, 2, 3, 3, 4, 5] .

?- add(0, [1,2,4,5], L).
L = [0, 1, 2, 4, 5].

?- add(6, [1,2,4,5], L).
L = [1, 2, 4, 5, 6] .

?- add(1,[],L).
```

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
L = [1, 2, 3, 3, 4, 5] .

?- add(0, [1,2,4,5], L).
L = [0, 1, 2, 4, 5].

?- add(6, [1,2,4,5], L).
L = [1, 2, 4, 5, 6] .

?- add(1,[],L).
L = [1] .
```

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
L = [1, 2, 3, 3, 4, 5] .

?- add(0, [1,2,4,5], L).
L = [0, 1, 2, 4, 5].

?- add(6, [1,2,4,5], L).
L = [1, 2, 4, 5, 6] .

?- add(1,[],L).
L = [1] .

?- add(1,[2],L).
```

Ordenação

```
add(X,[],[X]).
add(X,[H|T],[H|T2]) :- X > H, add(X,T,T2).
add(X,[H|T],[X,H|T]) :- X =< H.
```

```
?- add(3, [1,2,3,4,5], L).
L = [1, 2, 3, 3, 4, 5] .

?- add(0, [1,2,4,5], L).
L = [0, 1, 2, 4, 5].

?- add(6, [1,2,4,5], L).
L = [1, 2, 4, 5, 6] .

?- add(1,[],L).
L = [1] .

?- add(1,[2],L).
L = [1, 2].
```

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

- Se uma lista é vazia, ela está ordenada.
- Caso contrário, se ela contém um ou mais elementos, ordeno a cauda e adiciono a cabeça na cauda já ordenada.

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

```
?- ordenacao([3,2,1,4,5,6,2],L).
```

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

```
?- ordenacao([3,2,1,4,5,6,2],L).
L = [1, 2, 2, 3, 4, 5, 6] .
```

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

```
?- ordenacao([3,2,1,4,5,6,2],L).
L = [1, 2, 2, 3, 4, 5, 6] .

?- ordenacao([1],L).
```

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

```
?- ordenacao([3,2,1,4,5,6,2],L).
L = [1, 2, 2, 3, 4, 5, 6] .

?- ordenacao([1],L).
L = [1] .
```

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

```
?- ordenacao([3,2,1,4,5,6,2],L).
L = [1, 2, 2, 3, 4, 5, 6] .

?- ordenacao([1],L).
L = [1] .

?- ordenacao([],L).
```

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

```
?- ordenacao([3,2,1,4,5,6,2],L).
L = [1, 2, 2, 3, 4, 5, 6] .

?- ordenacao([1],L).
L = [1] .

?- ordenacao([],L).
L = [].
```

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

```
?- ordenacao([3,2,1,4,5,6,2],L).
L = [1, 2, 2, 3, 4, 5, 6] .

?- ordenacao([1],L).
L = [1] .

?- ordenacao([],L).
L = [].

?- ordenacao([1,6,5,4,3,2,1],L).
```

Ordenação

```
ordenacao([],[]).
ordenacao([H|T],LResultado) :-
    ordenacao(T,LResultadoCauda),
    add(H,LResultadoCauda,LResultado).
```

```
?- ordenacao([3,2,1,4,5,6,2],L).
L = [1, 2, 2, 3, 4, 5, 6] .

?- ordenacao([1],L).
L = [1] .

?- ordenacao([],L).
L = [].

?- ordenacao([1,6,5,4,3,2,1],L).
L = [1, 1, 2, 3, 4, 5, 6] .
```

Prolog

Alguns predicados prontos

- member(?Elemento, ?Lista)
- length(?Lista, ?Comprimento)
- append(?Lista1, ?Lista2, ?ListaContatenada)
- nth0(?Posicao, ?Lista, ?Elemento)
- list_to_set(?Lista, ?Conjunto)
- max_member(?Max, +Lista)
- min_member(?Max, +Lista)
- reverse(?Lista1, ?Lista2)
- permutation(?Lista1, ?Lista2)

Prolog - Alguns Links Úteis

- http://www.swi-prolog.org/pldoc/man?section=lists
- http://lpn.swi-prolog.org/lpnpage.php?pageid=online
- http://www.swi-prolog.org/pldoc/doc/_CWD_/index.html

Prolog

Ver atividade no Moodle