1. Listas

/\* o objectivo “?- heroi(X).”\*/

heroi(X) :- ajuda(X, Y), heroi(Y).

heroi(deadpool).

heroi(frodo).

ajuda(sam, frodo).

?- heroi(X).

X = sam ;

X = deadpool ;

X = frodo.

2.

/\* lista L vai ser escrita no ecrã, um elemento por linha\*/

escreveLista([]).

escreveLista([H | L]) :- format('H='),write(H), nl, format('L='),write(L), nl, escreveLista(L).

?- escreveLista([1, a, ola, 3]).

H=1

L=[a,ola,3]

H=a

L=[ola,3]

H=ola

L=[3]

H=3

L=[]

true.

/\*Versão iterativa\*/

multPorN(L1, N, L2) :- multPorN(L1, N, L2, []).

multPorN([], \_, L, L).

multPorN([H | T], N, L2, Ac) :-

H1 is H \* N,

append(Ac, [H1], Ac1),

multPorN(T, N, L2, Ac1).

?- trace.

true.

[trace] ?- multPorN([1,2], 1, L).

Call: (10) multPorN([1, 2], 1, \_12906) ? creep

Call: (11) multPorN([1, 2], 1, \_12906, []) ? creep

Call: (12) \_15056 is 1\*1 ? creep

Exit: (12) 1 is 1\*1 ? creep

Call: (12) lists:append([], [1], \_16690) ? creep

Exit: (12) lists:append([], [1], [1]) ? creep

Call: (12) multPorN([2], 1, \_12906, [1]) ? creep

Call: (13) \_19162 is 2\*1 ? creep

Exit: (13) 2 is 2\*1 ? creep

Call: (13) lists:append([1], [2], \_20796) ? creep

Exit: (13) lists:append([1], [2], [1, 2]) ? creep

Call: (13) multPorN([], 1, \_12906, [1, 2]) ? creep

Exit: (13) multPorN([], 1, [1, 2], [1, 2]) ? creep

Exit: (12) multPorN([2], 1, [1, 2], [1]) ? creep

Exit: (11) multPorN([1, 2], 1, [1, 2], []) ? creep

Exit: (10) multPorN([1, 2], 1, [1, 2]) ? creep

L = [1, 2].

/\*Versão recursiva\*/

multPorN1([], \_, []).

multPorN1([H | T], N, [H1 | T1]) :-

H1 is H \* N,

multPorN1(T, N, T1).

trace] ?- multPorN1([1,2], 1, L).

Call: (10) multPorN1([1, 2], 1, \_34926) ? creep

Call: (11) \_36260 is 1\*1 ? creep

Exit: (11) 1 is 1\*1 ? creep

Call: (11) multPorN1([2], 1, \_36262) ? creep

Call: (12) \_38706 is 2\*1 ? creep

Exit: (12) 2 is 2\*1 ? creep

Call: (12) multPorN1([], 1, \_38708) ? creep

Exit: (12) multPorN1([], 1, []) ? creep

Exit: (11) multPorN1([2], 1, [2]) ? creep

Exit: (10) multPorN1([1, 2], 1, [1, 2]) ? creep

L = [1, 2].

/\*Versão funcionais\*/

multPorN2(L1, N, L2) :-

maplist(mult(N), L1, L2).

mult(N, E, E1) :- E1 is E\*N.

maplist(P,L,M) succeeds by applying the predicate P to each element of a list L to form a new list M

3.

/\*predicado duplicaElem/2, tal que duplicaElem(Lst1,

Lst2) significa que a lista Lst2 é o resultado de repetir cada um dos elementos da

lista Lst1.\*/

/\*Versão iterativa\*/

duplicaElem(L1, L2) :-

duplicaElem(L1, L2, []).

duplicaElem([], L, L).

duplicaElem([H | T], L, Ac) :-

append(Ac, [H, H], Ac1),

duplicaElem(T, L, Ac1).

[trace] ?- duplicaElem([a,b], L).

Call: (10) duplicaElem([a, b], \_998) ? creep

Call: (11) duplicaElem([a, b], \_998, []) ? creep

Call: (12) lists:append([], [a, a], \_3154) ? creep

Exit: (12) lists:append([], [a, a], [a, a]) ? creep

Call: (12) duplicaElem([b], \_998, [a, a]) ? creep

Call: (13) lists:append([a, a], [b, b], \_5634) ? creep

Exit: (13) lists:append([a, a], [b, b], [a, a, b, b]) ? creep

Call: (13) duplicaElem([], \_998, [a, a, b, b]) ? creep

Exit: (13) duplicaElem([], [a, a, b, b], [a, a, b, b]) ? creep

Exit: (12) duplicaElem([b], [a, a, b, b], [a, a]) ? creep

Exit: (11) duplicaElem([a, b], [a, a, b, b], []) ? creep

Exit: (10) duplicaElem([a, b], [a, a, b, b]) ? creep

L = [a, a, b, b].

/\*Versão recursiva\*/

duplicaElem1([], []).

duplicaElem1([H | T], L) :-

duplicaElem1(T, Dup),

L = [H, H | Dup].

[trace] ?- duplicaElem1([a,b], L).

Call: (10) duplicaElem1([a, b], \_12966) ? creep

Call: (11) duplicaElem1([b], \_14294) ? creep

Call: (12) duplicaElem1([], \_15106) ? creep

Exit: (12) duplicaElem1([], []) ? creep

Call: (12) \_14294=[b, b] ? creep

Exit: (12) [b, b]=[b, b] ? creep

Exit: (11) duplicaElem1([b], [b, b]) ? creep

Call: (11) \_12966=[a, a, b, b] ? creep

Exit: (11) [a, a, b, b]=[a, a, b, b] ? creep

Exit: (10) duplicaElem1([a, b], [a, a, b, b]) ? creep

L = [a, a, b, b].

/\*Versão funcionais\*/

duplicaElem2(L1, L2) :-

findall([E, E], member(E, L1), LAux),

flatten(LAux, L2).

Findall(X,G,L) construct a list L consisting of all of the objects X such that gal G is satisfied.

?- flatten([[a, a], [b, b]],L).

L = [a, a, b, b].