CÓDIGOS PROLOG – MIGUEL SCHPEIR

UNIVERSIDAD NACIONAL DEL LITORAL

2011

```
/*
Dada una lista, escribir los predicados PROLOG necesarios para contabilizar las
apariciones de un determinado elemento en la lista.
Ejemplo: ?-aparicion(a,[a,b,c,a,b,a],X). % PROLOG contestaría X=3.
*/

count(_,[],0) :- !.
count(X,[Y|Z],Nro) :- X=Y, !, count(X,Z,Nro1), Nro is Nro1+1.
count(X,[_|Z],Nro) :- count(X,Z,Nro).
```

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

aplanar([],[]).
aplanar([X|C],L) :- not(X=[_|_]), !, aplanar(C,L1), concatenar([X],L1,L).
aplanar([X|C],L) :- aplanar(X,L1), aplanar(C,L2), concatenar(L1,L2,L).

eliminar(_, [], []) :- !.
eliminar(X, [X|L], L) :- !.
eliminar(X, [Y|L], [Y|L1]) :- eliminar(X, L, L1).
```

```
menor([X],X) :- !.

menor([X1,X2|C],L) :- X1=<X2, !, concatenar([X1],C,L1), menor(L1,L).

menor([_,X2|C],L) :- concatenar([X2],C,L1), menor(L1,L).


ord([],[]) :- !.

ord(X,[Menor|CL]) :- menor(X,Menor), eliminar(Menor,X,L2), ordenar(L2,CL).


ordenar(X,L) :- aplanar(X,L1), ord(L1,L).
```

```
% Conjuntos

pertenece(_,[]) :- fail.

pertenece(X,[X|_]).

pertenece(X,[_|C]) :- pertenece(X,C).


subconjunto([],_) :- true.

subconjunto([X|C],L) :- pertenece(X,L), subconjunto(C,L).


disjuntos([],_) :- true.

disjuntos([X|C],L) :- not(pertenece(X,L)), disjuntos(C,L).


% Interseccion

int(_,[],[]) :- !.

int([],_,[]) :- !.

int([X|C1],[X|C2],[X|C3]) :- int(C1,C2,C3), !.
```

```
int([X|C],[_|C2],L) :- pertenece(X,C2), !, int(C,C2,L3), append([X],L3,L).

int([_|C],L2,L) :- int(C,L2,L).


%int([a,b,c,d],[b,c,e,f],L). -> L=[b,c].

% FUNCIONA SOLO SI LA LISTA ESTA ORDENADA!


% Union

insertar(L,1,X,[X|L]):-!.

insertar([X|Y],N,R,[X|L]):-N1 is N-1, insertar(Y,N1,R,L).


union2([X|C],L) :- pertenece(X,L), !, union2(C,L).

union2([X|C],L) :- length(L,N), insertar(L,N,[X],L1), union2(C,L1).

union(L,L1,L) :- union2(L1,L).


%union([a,b,c,d],[d,e,f],L). -> L=[a,b,c,d,e,f].


% Inserta un elemento sin admitir duplicados

ins(X,[],[X]) :- !.

ins(X,L,L) :- pertenece(X,L), !.

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

```
% Verifica si dos elementos de una lista son consecutivos

consec(_,_,[]) :- fail, !.

consec(X1,X2,[X1,X2|_]) :- true, !.

consec(X1,X2,[_|C]) :- consec(X1,X2,C).
```

/*Dada una lista de letras, por ejemplo: L=[a,b,c,d,e,f,g], se pide:

a)Dado un elemento de una lista y la lista, devuelva una sublista que tendrá los elementos de la lista original

desde el elemento indicado (inclusive) hasta el final de la lista.

Ejemplo: desde(c,L,L1) -> L1=[c,d,e,f,g]. */

```
desde(_,[],[]) :- !.

desde(E,[E|C],[E|C]) :- !.

desde(E,[_|C],L) :- desde(E,C,L).
```

/*b) Dado un elemento de una lista y la lista, devuelva una sublista que tendrá los elementos de la lista original

desde el primer elemento de la lista HASTA el elemento indicado (inclusive).

Ejemplo: hasta(d,L,L1) -> L1=[a,b,c,d]. */

```
hasta(_,[],[]) :- !.

hasta(E,[E|_],[E|[]]) :- !.

hasta(E,[X|C],[X|C1]) :- hasta(E,C,C1).
```

/*c) Utilizando "desde" y "hasta", dados dos elementos de una lista y la lista devuelva una

sublista que tendrá los elementos de la lista original desde el primer elemento indicado

hasta el segundo elemento indicado (ambos inclusive).

Ejemplo: desde_hasta(c,d,L,L1) -> L1=[c,d]. */

```
pertenece(_,[]) :- fail.
```

```
pertenece(X,[X|_]).

pertenece(X,[_|C]) :- pertenece(X,C).


int(_,[],[]) :- !.

int([],_,[]) :- !.

int([X|C1],[X|C2],[X|C3]) :- int(C1,C2,C3), !.

int([X|C],[_|C2],L) :- pertenece(X,C2), !, int(C,C2,L3), append([X],L3,L).

int([_|C],L2,L) :- int(C,L2,L).


desde_hasta(_,_,[],[]) :- !.

desde_hasta(X,X,_,[X]) :- !.

desde_hasta(X,Y,L1,L) :- desde(X,L1,Ld), hasta(Y,L1,Lh), int(Ld,Lh,L).
```

/*

Dado un grafo dirigido, una representación para el mismo en Prolog podría consistir en una lista de listas, donde cada sublista contiene como elementos un nodo y una lista de los nodos hacia los cuales éste está conectado, por ejemplo:

(A) [[a, [b]], [b, [c, d]], [c, [e]], [d, [a, e]], [e, [a]]]

Otra representación válida consiste en una lista que contenga dos sublistas: una representando el conjunto de nodos (o vértices) y otra representando el conjunto de arcos, donde cada arco es a su vez una lista de dos elementos, el nodo inicial y el final. Por ejemplo:

(B) [[a, b, c, d, e], [[a, b], [b, c], [b, d], [c, e], [d, a], [d, e], [e, a]]]

Escriba un programa en prolog que permita pasar del formato A al B

Ej:

transformar([[a, [b]], [b, [c, d]], [c, [e]], [d, [a, e]], [e, [a]]], X).

X = [[a, b, c, d, e], [[a, b], [b, c], [b, d], [c, e], [d, a], [d, e], [e, a]]].

*/

transformar(Grafo,[Nodos,Arcos]) :- trans_armar(Grafo,Nodos,Arcos).

trans_armar([],[],[]) :- !.

trans_armar([[Nodo,Conectado]|C],[Nodo|C1],Arcos):-
trans_arcos(Nodo,Conectado,ArcosNodo),trans_armar(C,C1,A),append(ArcosNodo,A,Arcos).

trans_arcos(_N,[],[]) :- !.

trans_arcos(N,[C|C1],[[N,C]|C2]):- trans_arcos(N,C1,C2).

% Eliminar los últimos 3 elementos de una lista

% Borrar los ultimos 3 elementos

elimn(1,[_|Lc], Lc).

elimn(N,L,_) :- length(L,S), N>S, fail, !.

elimn(N,[Lx|Lc], Lmod) :- N1 is N-1, elimn(N1, Lc, L1), Lx=[_], !, append(Lx,L1,Lmod), !.

elimn(N,[Lx|Lc], Lmod) :- N1 is N-1, elimn(N1, Lc, L1), append([Lx],L1,Lmod), !.

elim3([],[]) :- !.

elim3(L,LMod) :- length(L,N), elimn(N,L,Lmod).

/*Cree un predicado en Prolog que dada una lista de números recibida en el primer parámetro,

unifique un segundo parámetro con los números en posiciones impares y en un tercero los números en posiciones pares.

Ej.:

?- sumar([1, 2, 3, 4, 5], X, Y).

X = 9

```
Y = 6

*/

sum([],0) :- !.

sum([X|C],S) :- sum(C,S1), S is S1+X.

formarlista([],[],[]) :- !.

formarlista([X],[X],[]) :- !.

formarlista([X,Y|C],[X|Li],[Y|Lp]) :- formarlista(C,Li,Lp).

sumar([],0,0) :- !.

sumar(L,Si,Sp) :- formarlista(L,Li,Lp), sum(Li,Si), sum(Lp,Sp).
```

```
/* Elimina el elemento x de la lista en todos los niveles*/

borra(_,[],[]) :- !.

borra(X,[X],[]) :- !.

borra(X,[X|C],L) :- borra(X,C,L), !.

borra(X,[Y|C],L) :- Y=[_|_], !, borra(X,Y,L1), borra(X,C,L2), append([L1],L2,L).

borra(X,[Y|C],L) :- borra(X,C,L1), append([Y],L1,L).

% borra(1,[[1,2],2,3],L).
```

```
% Elimina el elemento parámetro de la lista

eliminar(_,[],[]) :- !.

eliminar(X,[X|L],L) :- !.

eliminar(X,[Ca|Co], [Ca|L]) :- eliminar(X,Co,L).
```

```
factorial(0,1).

factorial(1,1).
```

factorial(X,Y) :- X1 is X-1, factorial(X1,Y1), Y is X*Y1.

&factorial x = x * factorial (x-1)

% Dado un grafo, determina el camino a seguir para dibujarlo sin levantar el lápiz (grafo plano)

arista(a, b).

arista(a, c).

arista(a, e).

arista(a, d).

arista(b, c).

arista(c, d).

arista(c, e).

arista(d, e).

conectado(X, Y) :- arista(X, Y); arista(Y, X).

aristas(L) :- findall([X, Y], conectado(X, Y), L).

nodos(L) :- findall(X, conectado(X, _), L1), findall(Y, conectado(_, Y), L2), append(L1, L2, L3), sort(L3, L).

% Para no pasar dos veces por la misma arista

borrar(_, [], []) :- !.

borrar([X, Y], [[X1, Y1]|R], R1) :- (X = X1, Y = Y1; Y = X1, X = Y1), !, borrar([X, Y], R, R1).

borrar([X, Y], [[X1, Y1]|R], [[X1, Y1]|R1]) :- borrar([X, Y], R, R1).

recorrido(R) :- aristas(La),nodos(Ln), member(X, Ln), camino(X, La, R).

camino(X, [], [X]) :- !.

camino(X, L1, [X | R]) :- conectado(X, Y), member([X, Y], L1), borrar([X, Y], L1, L), camino(Y, L, R).

% Determina si dos listas son iguales en todos los niveles

```
iguales([],[]) :- !.

iguales([X|Y],[X|Z]) :- iguales(Y,Z), !.

iguales([X|C],[Y|C1]) :- X=[_|_], Y=[_|_], !, iguales(X,Y), iguales(C,C1), !.

% iguales([1,2,[1,3,2,4],[5]], [1,2,[1,3,2,4],[5]]). -> true.
```

```
% Determina si una palabra es o no un palíndromo

inversa([],[]) :- !.

inversa([X|C],L) :- inversa(C,L1), append(L1,[X],L).

palindromo([],[]).

palindromo(P1,P2) :- atom_chars(P1,Pa1), atom_chars(P2,Pa2), inversa(Pa1,Pa2).
```

```
% Minimo y maximo


minimo([X],X) :- !.

minimo([X1,X2|C],Min) :- X1<X2, !, minimo([X1|C],Min).

minimo([_,X2|C],Min) :- minimo([X2|C],Min).


maximo([X],X) :- !.

maximo([X1,X2|C],Max) :- X1>X2, !, maximo([X1|C],Max).

maximo([_,X2|C],Max) :- maximo([X2|C],Max).


max([],[]) :- !.

max([X],[X]) :- !.

max([X|Y],Max) :- max(Y,Max1), X>=Max1, !, Max is X.

max([_|Y],Max) :- max(Y,Max1), Max is Max1.
```

```
% Cuenta la cantidad de veces que ocurre un elemento en una lista en todos los niveles

ocurrencia(_,[],0) :- !.

ocurrencia(X,[X|C],S) :- ocurrencia(X,C,S1), S is S1+1, !.

ocurrencia(X,[Y|C],S) :- Y=[_|_], !, ocurrencia(X,Y,S1), ocurrencia(X,C,S2), S is S1+S2.

ocurrencia(X,[_|C],S) :- ocurrencia(X,C,S).

% ocurrencia(1, [1,2,[1,7,1],1],S).
```

```
/* Escribir un programa en Prolog que aplane una lista y la ordene. El predicado ordenar/2 recibe

una lista cuyos elementos pueden ser otras listas, los elementos individuales de las listas deben ser

números enteros, si se encontrara un elemento que no sea numérico, el predicado debería fallar.

Para ordenar y aplanar la lista se deben construir predicados que lo hagan, no usar predefinidos si

los hubiere.*/


cat([],L,L).

cat([L|L1],L2,[L|L3]) :- cat(L1,L2,L3).


minimo([X], X) :- !.

minimo([X1, X2 | L], X) :- X1 @=< X2 -> minimo([X1|L], X); minimo([X2|L], X).


aplanar([],[]) :- !.

aplanar([X|Y],Lap) :- X=[_|_], !, aplanar(X,L1ap), aplanar(Y,L2ap), cat(L1ap,L2ap,Lap).

aplanar([X|Y],Lap) :- aplanar(Y,L1ap), cat([X],L1ap,Lap).
```

```
borrar([],_,[]).

borrar([X|C],M,C):- X==M,!.

borrar([X|C],M,L):- borrar(C,M,L2), cat([X],L2,L).


ordenar([],[]) :- !.

ordenar(L,[Min|Y]) :- minimo(L,Min), borrar(L,Min,L1), ordenar(L1,Y).
```

```
% Parcial 21/05/2010

% Ejercicio 1-1

palindromo([]):- !, fail.

palindromo([_]):- !, fail.

palindromo(L):- invertir(L, L1), L = L1.



% Ejercicio 1-2

concatenar([], L, L).

concatenar([X|L1], L2, [X|L3]) :- concatenar(L1, L2, L3).


invertir([X|L], L1) :- X = [_|_], !, invertir(X, X1), invertir(L, L2), concatenar(L2, [X1], L1).

invertir([], []):-!.

invertir([X|L], L1) :- invertir(L, L2), concatenar(L2, [X], L1).
```

```
split_c1([], _, R1, R2, [R1|R2]).

split_c1([X|L], X, R1, R2, R) :- !, split_c1(L, X, [], [R1|R2], R).

split_c1([X|L], Y, R1, R2, R) :- split_c1(L, Y, [X|R1], R2, R).

split_c(L, S, R) :- split_c1(L, S, [], [], R1), invertir(R1, R).



% Ejercicio 1-3

%arbol_valido([], _) :- !.

%arbol_valido([nil | L], N) :- !, arbol_valido(L, N).

%arbol_valido([X | L], N) :- t(I, R, D) = X, not(member(R, N)), arbol_valido([I, D | L], [R | N]).

prof_arbol(T, Prof) :- arbol_valido([T], []), mas_prof([T], 0, Prof).

mas_prof([], Nivel, Nivel) :- !.

mas_prof(L, Nivel, Prof) :-
                    Nivel1 is Nivel + 1,
                    proximo_nivel(L, L1),
                    mas_prof(L1, Nivel1, Prof).

%proximo_nivel([], []):- !.

%proximo_nivel([ t( nil, _, nil ) | L ], L1) :- !, proximo_nivel(L, L1).

%proximo_nivel([ t( nil, _, D ) | L ], [ D | L1 ]) :- !, proximo_nivel(L, L1).
```

```
%proximo_nivel([ t( I, _, nil ) | L ], [ I | L1 ]) :- !, proximo_nivel(L, L1).

%proximo_nivel([ t( I, _, D ) | L ], [ I, D | L1 ]) :- proximo_nivel(L, L1).



% Ejercicio 2-1

valida_dup([], []) :- !.

valida_dup([X], [X]) :- !.

valida_dup([X, X | L], L1) :- !, valida_dup([X | L], L1).

valida_dup([X, Y | L], [X | L1]) :- !, valida_dup([Y | L], L1).



% Ejercicio 2-2

split_n1([], _, _, R2, [R2]) :- !.


split_n1(L, N, N, R2, [R3|R]) :- !, invertir(R2, R3), split_n1(L, N, 0, [], R).


split_n1([X|L], N, N1, R2, R) :- !, N2 is N1 + 1, split_n1(L, N, N2, [X|R2], R).


split_n(L, N, R) :- N > 0, split_n1(L, N, 0, [], R).



% Ejercicio 2-3

arbol_valido([], _) :- !.

arbol_valido([nil | L], N) :- !, arbol_valido(L, N).
```

```
arbol_valido([X | L], N) :- t(I, R, D) = X, not(member(R, N)), arbol_valido([I, D | L], [R | N]).

ancho_arbol(T, Nivel, Ancho) :- arbol_valido([T], []), mas_ancho([T], 1, 1, 1, Nivel, Ancho).

mas_ancho([], _, Nivel, Ancho, Nivel, Ancho) :- !.

mas_ancho(L, Na, N, A, Nivel, Ancho) :-
                Na1 is Na + 1,
                proximo_nivel(L, L1),
                length(L1, A1),
                (A1 > A -> A2 = A1, N2 = Na1; A2 = A, N2 = N),
                mas_ancho(L1, Na1, N2, A2, Nivel, Ancho).

proximo_nivel([], []):- !.
proximo_nivel([ t( nil, _, nil ) | L ], L1) :- !, proximo_nivel(L, L1).
proximo_nivel([ t( nil, _, D ) | L ], [ D | L1 ]) :- !, proximo_nivel(L, L1).
proximo_nivel([ t( I, _, nil ) | L ], [ I | L1 ]) :- !, proximo_nivel(L, L1).
proximo_nivel([ t( I, _, D ) | L ], [ I, D | L1 ]) :- proximo_nivel(L, L1).
```

```
/*a) Escribir un predicado PROLOG para que dada una lista de números, determine el
número de elementos positivos (el 0 se considera como positivo) y el número de
elementos negativos.
Ejemplo: pos_y_neg([3,-1,9,0,-3,-5,-8],3,4). % PROLOG respondería sí. */

pos_y_neg([],0,0).
```

```
pos_y_neg([X|C],P,N) :- X>=0, !, pos_y_neg(C,P1,N), P is P1+1.

pos_y_neg([_|C],P,N) :- pos_y_neg(C,P,N1), N is N1+1.


/*b) Escribir un predicado PROLOG para que dada una lista de números, construya

dos listas, una de números positivos y otra de negativos y que determine la

cardinalidad de ambas listas.

Ejemplos:

pos_y_neg([3,-1,9,0,-3,-5,-8],3,[3,9,0],4,[-1,-3,-5,-8]). % PROLOG respondería sí.

pos_y_neg([3,-1,9,0,-3,-5,-8],Npos,Pos,Nneg,Neg).  %PROLOG respondería: Npos=3, Pos=[3,9,0],
Nneg=4, Neg=[-1,-3,-5,-8] */


pos_y_neg2([],0,[],0,[]).

pos_y_neg2([X|C],Npos,Lpos,Nneg,Lneg) :- X>=0, !, pos_y_neg2(C,Npos1,Lpos1,Nneg,Lneg), Npos
is Npos1+1, append([X],Lpos1,Lpos).

pos_y_neg2([X|C],Npos,Lpos,Nneg,Lneg) :- pos_y_neg2(C,Npos,Lpos,Nneg1,Lneg1), Nneg is
Nneg1+1, append([X],Lneg1,Lneg).
```

```
/*Ejercicio 1.10 Definir la relación selecciona(?X,?L1,?L2) que se verifique si L2 es la lista obtenida

eliminando una ocurrencia de X en L1 */

%selecciona(_,[],[]) :- !.    <--- Solo para validar

selecciona(X,[X|C1],C1) :- !.

selecciona(X,[L1|C1],[L1|L3]) :- selecciona(X,C1,L3).

%selecciona(3,[1,2,3,4,5,6],L). -> L=[1,2,4,5,6].
```

```
semejanza([],[],0) :- !.

semejanza([],X,S) :- !, length(X,X1), S is -X1.

semejanza(X,[],S) :- !, length(X,X1), S is -X1.
```

```
semejanza([X|C1],[X|C2],S) :- !, semejanza(C1,C2,S1), S is S1+1.

semejanza([_|C1],[_|C2],S) :- !, semejanza(C1,C2,S1), S is S1-1.


/*Para el desarrollo de ésta segunda parte del programa, se puede usar el predicado predefinido

“atom_chars(C, L)” que toma un átomo (C) y lo transforma en una lista de literales (L):

atom_chars(hola, L). --> L = [h, o, l, a].

La definición del diccionario en principio es:

dic([sanar, hola, sabana, sabalo, prueba, computadora, cartera, mate, termo, mesa, silla, sarna]).

buscar(hola, L). --> L = [hola]

buscar(holo, L). --> L = [[hola, 2]]

buscar(saban, L). --> L = [[sanar, 1], [sabana, 4], [sabalo, 2]]

*/


dic([sanar, hola, sabana, sabalo, prueba, computadora, cartera, mate, termo, mesa, silla, sarna]).
```

```
% Forma una sublista desde-hasta donde se especifique


pertenece(_,[]) :- fail.

pertenece(X,[X|_]).

pertenece(X,[_|C]) :- pertenece(X,C).


desde(_,[],[]) :- !.

desde(X,[X|L],[X|L]) :- !.

desde(X,[_|L],Lmod) :- desde(X,L,Lmod).
```

```
hasta(_,[],[]) :- !.

hasta(E,[E|_],[E|[]]) :- !.

hasta(E,[X|C],[X|C1]) :- hasta(E,C,C1).


int(_,[],[]) :- !.

int([],_,[]) :- !.

int([X|C1],[X|C2],[X|C3]) :- int(C1,C2,C3), !.

int([X|C],[_|C2],L) :- pertenece(X,C2), !, int(C,C2,L3), append([X],L3,L).

int([_|C],L2,L) :- int(C,L2,L).


sublista(D,H,L1,L) :- desde(D,L1,LD), hasta(H,L1,LH), int(LD,LH,L).
```

```
% Ejercicio 1.12 Definir la relación sublista(?L1,?L2) que se verifique si L1 es una sublista de L2.

% L1 es un par


sublista([X,Y],[X,Y|_]) :- !.

sublista([X,Y],[_|C]) :- sublista([X,Y],C).


% Ejercicio 1.17 De?nir la relación subconjunto(+L1,?L2) que se veri?que si L1 es un subconjunto
de L2


pertenece(_,[]) :- !.

pertenece(X,[X|_]) :- !.

pertenece(X,[_|C]) :- pertence(X,C).


% SOLO si L1 es un par
```

```
subconjunto([],_) :- !.
subconjunto([X,Y],L) :- pertenece(X,L),pertenece(Y,L).

% Si L1 NO es un par
subconjunto([],_) :- !.
subconjunto([X|C],L) :- pertenece(X,L),subconjunto(C,L).
```

```
/*Definir la relación
trasladar(L1,L2)
que permita trasladar una lista de números (L1) a una lista de sus correspondientes
nombres (L2).
Ejemplo: trasladar([1,2,3,7],L) -> L=[uno,dos,tres,siete] */

significa(0,cero).
significa(1,uno).
significa(2,dos).
significa(3,tres).
significa(4,cuatro).
significa(5,cinco).
significa(6,seis).
significa(7,siete).
significa(8,ocho).
significa(9,nueve).

trasladar([],[]).
```

```
trasladar([X|C],L) :- significa(X,Num), trasladar(C,L1), append([Num],L1,L).
```

```
ultimo([],[]) :- !.

ultimo([X],X) :- !.

ultimo([_|C],E) :- ultimo(C,E).


penultimo([],[]) :- !.

penultimo([X,_],X) :- !.

penultimo([_|C],E) :- penultimo(C,E).
```

/*Dadas dos listas de números ordenadas en orden creciente, calcular la unión ordenada de las dos listas

iniciales. La solución debe basarse en la característica de que las listas originales están ordenadas.

Ejemplo: Dadas L1=[1,5,7,9], L2=[4,5,6,9,10], union_ordenada(L1,L2,L3). -> L3=[1,4,5,6,7,9,10].*/

```
%un_ord([1,2,3,4,5],[2,3,5,7,9],L).


un_ord(L,[],L) :- !.

un_ord([],L,L) :- !.

un_ord([X|C],[Y|C1],L) :- X<Y, !, un_ord(C,[Y|C1],L1), append([X],L1,L).

un_ord([X|C],[Y|C1],L) :- X=Y, !, un_ord(C,C1,L1), append([X],L1,L).

un_ord(L1,[Y|C1],L) :- un_ord(L1,C1,L2), append([Y],L2,L).


% Nueva, mejor:
```

```
unord([],[],[]) :- !.

unord(L,[],L) :- !.

unord([],L,L) :- !.

unord([X|Y],[X|Z],[X|L]) :- unord(Y,Z,L), !.

unord([X|Y],[W|Z],[X|L]) :- X<W, !, unord(Y,[W|Z],L).

unord([X|Y],[W|Z],[W|L]) :- unord([X|Y],Z,L).
```

```
% ------------------------------------------------------------------

% EJERCICIOS DE PROGRAMACION LOGICA: PROLOG.

% ------------------------------------------------------------------

% Esta linea no la tengais en cuenta (es para que no aparezcan

% mensajes WARNING)

:- set_prolog_flag(multi_arity_warnings,off).

% ------------------------------------------------------------------


% ------------------------------------------------------------------

% Introduccion. Relaciones familiares.

% ------------------------------------------------------------------

padre_de(juan, pedro).

padre_de(juan, maria).

padre_de(pedro, miguel).

madre_de(maria, david).


abuelo_de(L,M) :- padre_de(L,K),

                  padre_de(K,M).
```

```
abuelo_de(X,Y) :- padre_de(X,Z),
                  madre_de(Z,Y).

% Consultas: abuelo_de(juan,X). % De quien es Juan abuelo?
%           abuelo_de(X,Y).    % Relaciones abuelo/nieto.

% -----------------------------------------------------------------
% Introduccion. Animales domesticos.
% -----------------------------------------------------------------
animal_domestico(X) :- animal(X), ladra(X).
animal_domestico(X) :- animal(X), hace_burbujas(X).

animal(spot).
animal(barry).
animal(hobbes).

ladra(spot).
hace_burbujas(barry).
ruge(hobbes).

% -----------------------------------------------------------------
% Programacion recursiva.
% -----------------------------------------------------------------
progenitor(X,Y) :- padre_de(X,Y).
progenitor(X,Y) :- madre_de(X,Y).
```

```
antepasado(X,Y) :- progenitor(X,Y).
antepasado(X,Y) :- progenitor(X,Z),
                   antepasado(Z,Y).

parientes(X,Y) :-
          antepasado(Z,X),
          antepasado(Z,Y).

primos(X,Y) :-
          progenitor(X1,X),
          progenitor(Y1,Y),
          progenitor(Z,X1),
          progenitor(Z,Y1).

misma_generacion(X,Y) :-
          progenitor(Z,X),
          progenitor(Z,Y).
misma_generacion(X,Y) :-
          progenitor(X1,X),
          progenitor(Y1,Y),
          misma_generacion(X1,Y1).

% ---------------------------------------------------------------
% Datos Estructurados
% ---------------------------------------------------------------
raiz(arbol(Raiz,_,_),Raiz).
```

```
hoja_izquierda(arbol(Hoja,void,void),Hoja).
hoja_izquierda(arbol(_,Izq,_),Hoja):-
        hoja_izquierda(Izq,Hoja).
hoja_izquierda(arbol(_,void,Dcha),Hoja):-
        hoja_izquierda(Dcha,Hoja).

% Consulta: raiz(X,a), hoja_izquierda(X,b).

% Ejemplo de arbol.
arbol1(arbol(a,arbol(b,void,void),
            arbol(c,arbol(d,void,void),
                  arbol(e,void,void)))).

% ---------------------------------------------------------------
% Datos Estructurados. Recorridos de Arboles.
% (aunque no lo hemos visto expresamente, es interesante ver como
% se recorren los arboles binarios)
% ---------------------------------------------------------------
inorden(void).
inorden(arbol(Nodo,Izq,Der)) :-
        inorden(Izq),
        display(Nodo),
        inorden(Der).

preorden(void).
```

```
preorden(arbol(Nodo,Izq,Der)) :-
        display(Nodo),
        preorden(Izq),
        preorden(Der).

postorden(void).
postorden(arbol(Nodo,Izq,Der)) :-
        postorden(Izq),
        postorden(Der),
        display(Nodo).

% ----------------------------------------------------------------
% Programacion recursiva. Listas II
% ----------------------------------------------------------------

iguales(X,X).

% consultas: iguales([a,b],[a|X]).
%         iguales([a],[a|X]).
%         iguales([a],[a,b|X]).
%         iguales([],[X]).

% consultas: [a,b] = [a|X].
%         [a] = [a|X].
%         X = a.
%         [X,a] = [Y|X] %Esta consulta hay que pensarla despacio!
```

```
% ----------------------------------------------------------------

% Programacion recursiva. Listas III

% ----------------------------------------------------------------


% Definicion de lista.

list([]).

list([_|Xs]):-list(Xs).


% Pertenencia de un elemento a una lista.

member(X,[X|_]).

member(X,[_|Xs]) :- member(X,Xs).



% prefix(X,Y): tiene exito si la lista X es prefijo

%            de la lista Y.

prefix([],Ys) :-

           list(Ys).

prefix([X|Xs],[X|Ys]) :-

           prefix(Xs,Ys).


% suffix(X,Y): tiene exito si la lista X es sufijo

%            de la lista Y.

suffix(Xs,Xs) :-

           list(Xs).

suffix(Xs,[_|Ys]):-
```

```
        suffix(Xs,Ys).


% sublist(X,Y):tiene exito si la lista X es sublista
%           de la lista Y.
sublist(Xs,Ys) :-
        prefix(Xs,Ys).
sublist(Xs,[_|Ys]) :-
        sublist(Xs,Ys).


% ---------------------------------------------------------------
% Programacion recursiva. Listas IV
% ---------------------------------------------------------------
append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]) :-
        append(Xs,Ys,Zs).


% Consulta: append(X,Y,[a,b]).


reverse([],[]).
reverse([X|Xs],Ys) :-
        reverse(Xs,Zs),
        append(Zs,[X],Ys).


% Otra solucion mas eficiente para reverse, porque no recorre la
% lista tantas veces como lo hace append en la anterior.
% IMPORTANTE: utiliza DOS predicados reverse2, uno de aridad 2, y
```

```
%           el otro de aridad 3.
reverse2(Xs,Ys) :-
        reverse2(Xs,[],Ys).

reverse2([],Ys,Ys).
reverse2([X|Xs],Ys,Zs) :-
        reverse2(Xs,[X|Ys],Zs).

% Vuelta a los ejemplos de Listas III.
% Uso de append/3 para otras definiciones hechas anteriormente.
prefix2(Xs,Ys) :-
        append(Xs,_,Ys).

suffix2(Xs,Ys) :-
        append(_,Xs,Ys).

sublist2(Xs,Ys) :-
        append(_,Xs,B),
        append(B,_,Ys).

% Cuidado con la siguiente consulta:
%       sublist2(X,[a,b]). % No termina!!! ¿Por qu'e?

% ---------------------------------------------------------------
% Aritmetica I
% ---------------------------------------------------------------
```

```
% Probar las siguientes consultas:
%  display(3*4).
%  X is 3*4, display(X).
% Por que los resultados son distintos?


% ---------------------------------------------------------------
% Aritmetica II
% ---------------------------------------------------------------
% fibonacci(N,X): Tiene exito si X es el numero de
%                 Fibonacci de N.
fibonacci(0,0).
fibonacci(1,1).
fibonacci(N,X):-
        N > 1,
        N1 is N-1,
        N2 is N-2,
        fibonacci(N1,X1),
        fibonacci(N2,X2),
        X is X1+X2.


% length(L,N): Tiene exito si N es la longitud
%              de la lista X.
length([],0).
length([_|Xs],N) :-
        length(Xs,N1),
```

```
        N is N1 + 1.


% nodes(A,N):  Tiene exito si N es el numero de nodos
%             del arbol A.
nodes(void,0).
nodes(arbol(_,Izq,Der),N) :-
        nodes(Izq,N1),
        nodes(Der,N2),
        N is N1 + N2 + 1.


% ------------------------------------------------------------------
% Operador de poda: Tipos de corte I
% ------------------------------------------------------------------
% Ejemplo de corte blanco
max(X,Y,X) :- X > Y, !.
max(X,Y,Y) :- X =< Y.


% Ejemplo de corte verde
membercheck(X,[X|_]) :- !.
membercheck(X,[_|Xs]) :- membercheck(X,Xs).


% ------------------------------------------------------------------
% Operador de poda: Tipos de corte II
% ------------------------------------------------------------------


% Ejemplo de cortes rojos:
```

```
maxr(X,Y,X):- X > Y, !.

maxr(_X,Y,Y).



% parientes2(X,Y): Cierto si X e Y son parientes,

%               pero devolviendo una sola

%               solucion.

parientes2(X,Y) :- parientes(X,Y), !.



% arbol_check(X,Y):Cierto si X es un nodo del

%               arbol Y, pero que proporcione

%               un solo resultado.

arbol_check(X,arbol(X,_,_)) :- !.

arbol_check(X,arbol(_,I,_)) :-

        arbol_check(X,I), !.

arbol_check(X,arbol(_,_,D)):-

        arbol_check(X,D),!.

% ---------------------------------------------------------------

% Predicados meta-logicos II

% ---------------------------------------------------------------

length2(Xs,N):-

   var(Xs), integer(N), length_num(N,Xs).
```

```
length2(Xs,N):-
   nonvar(Xs), length_list(Xs,N).

length_num(0,[]).
length_num(N,[_|Xs]):-
   N > 0, N1 is N - 1, length_num(N1,Xs).

length_list([],0).
length_list([_|Xs],N):-
   length_list(Xs,N1), N is N1 + 1.

% Ejercicios. Definid los siguientes predicados:
% lista_ground(X): tiene exito si X es una lista de terminos
%              ground (sin variables).
lista_ground(Xs) :- ground(Xs), list(Xs).


% arbol_ligado(A,T): dado un arbol A no completamente instanciado
%              (con nodos o subárboles representados
%              mediante variables libres), debe instanciar
%              completamente A con el termino T (dejando un
%              arbol bien formado).
arbol_ligado(Nodo,T) :-  %Importante!: esta debe ser la PRIMERA clausula.
          var(Nodo),
          !,
          Nodo = arbol(T,void,void).
```

```
arbol_ligado(void,_):- !.


arbol_ligado(arbol(Nodo,Izq,Der),T) :-
        var(Nodo),
        !,
        Nodo = T,
        arbol_ligado(Izq,T),
        arbol_ligado(Der,T).


arbol_ligado(arbol(_,Izq,Der),T) :-
        !,
        arbol_ligado(Izq,T),
        arbol_ligado(Der,T).


% Ejemplo para arbol_ligado/2.
arbol2(arbol(a,arbol(b,void,void),
            arbol(_C,arbol(d,void,void),
                  _D))).


% Si se realiza la consulta arbol2(A),arbol_ligado(A,w).,
% debe dejar A instanciado de la siguiente forma:
% A = arbol(a,arbol(b,void,void),
%           arbol(w,arbol(d,void,void),
%                 arbol(w,void,void)))
```

```
% ----------------------------------------------------------------
% Meta-predicados: orden superior II
% ----------------------------------------------------------------

map(_,[],[]).
map(Pred,[X|Xs],[Y|Ys]) :-
        clone(Pred,Pred1),    % IMPORTANTE: es necesario hacer una
        arg(1,Pred1,X),       %           copia del predicado Pred.
        arg(2,Pred1,Y),
        call(Pred1),
        map(Pred,Xs,Ys).

clone(T1,T2) :-
        functor(T1,Nombre,Aridad),
        functor(T2,Nombre,Aridad).

% ----------------------------------------------------------------
% Modificacion dinamica de programas I
% ----------------------------------------------------------------
:- dynamic cont/1.

contador(X) :-
   retract(cont(X1)),
   X is X1 + 1,
   assert(cont(X)).
```

```
contador(1) :-
   assert(cont(1)).

% ----------------------------------------------------------------
% Modificacion dinamica de programas III
% ----------------------------------------------------------------
:- use_module(library(read)).
:- dynamic dict/2.
% Lectura de los datos de un fichero. El fichero debe tener hechos con
% la forma dict(Palabra,Significado), siendo ambos argumentos átomos
% Prolog.
leer_dic(F) :-
        open(F,read,Stream),
        leer_datos(Stream),
        close(Stream).

leer_datos(Stream) :-
        read(Stream,T),
        T \== end_of_file,
        assert(T),
        !,
        leer_datos(Stream).
leer_datos(_).

% Busqueda del significado de una palabra en el diccionario.
buscar(P,S) :-
```

```
	dict(P,S).


% Busqueda de todos los significados de una palabra en el diccionario.
% Como la forma de buscarlos en dict(P,S) es utilizando backtracking,
% para guardar los cambios en la lista de significados necesitamos
% utilizar el predicado dinamico todos/1.
:- dynamic todos/1.
buscar_todos(P,_) :-
	assert(todos([])),
	dict(P,S),
	retract(todos(L1)),
	append(L1,[S],L2),
	assert(todos(L2)),
	fail.


buscar_todos(_,L) :-
	retract(todos(L)).


% Consultas: leer_dic('ejercicios.txt').
%	buscar(hola,X).
%	buscar_todos(hola,X).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 1:
% Supongamos que tenemos el siguiente conocimiento sobre
```

```
% divisibilidad:
%   " 2 divide a 6"
%   " 2 divide a 12"
%   " 3 divide a 6"
%   " 3 divide a 12"
%   " Si un número es divisible por 2 y por 3 entonces es
%     divisible por 6"
%
% Escribir un programa que represente este conocimiento y usarlo
% para responder a las siguientes preguntas:
% (1) ¿Existe algún múltiplo de 2?
% (2) ¿Cuáles son los divisores de 6?
% (3) ¿Conocemos algún múltiplo de 6?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

divide(2,6).
divide(2,12).
divide(3,6).
divide(3,12).
divide(6, X):-
        divide(2, X),
        divide(3, X).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

% Ejercicio 2: En este ejercicio y los siguientes vamos a

% realizar un ejemplo que describe la carta de un restaurante.

% Los objetos que interesan son los platos que se pueden consumir

% y una primera clasificación puede ser la siguiente:

% * Entradas: paella, gazpacho, consomé

% * Carne: filete de cerdo, pollo asado

% * Pescado: trucha, bacalao

% * Postre: flan, nueces con miel, naranja

%

% Escribe como programa Prolog la clasificación de comidas

% del restaurante. El programa constara de 10 cláusulas y los

% predicados a usar son entrada/1, carne/1, pescado/1 y postre/1. % Prueba después el programa con preguntas como

% ?- carne(X).

% ?- carne(X), postre(X).

% ... etc...

%%

```
entrada(paella).

entrada(gazpacho).

entrada(consome).

carne(filete_de_cerdo).

carne(pollo_asado).

pescado(trucha).

pescado(bacalao).
```

postre(flan).

postre(nueces_con_miel).

postre(naranja).

%%%

% Ejercicio 3: Definir la relación "plato_principal(X)" que

% indicara que un plato principal es un plato de carne o de

% pescado.

%%%

```
plato_principal(X):-
        carne(X).
plato_principal(X):-
        pescado(X).
```

%%%

% Ejercicio 4: Definir la relación "comida(X,Y,Z)" que indicara

% que la comida consta de tres platos, una entrada "X", un plato

% principal "Y" y un postre "Z".

% Pidiéndole respuestas sucesivas a la pregunta ?- comida(X,Y,Z).

% podemos generar todas las posibles comidas del restaurante.

%%%

```
comida(X, Y, Z):-
	entrada(X),
	plato_principal(Y),
	postre(Z).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 5:
% (a) ¿Cómo se pregunta por las comidas con pescado sin modificar
%     el programa? comida(X, Y, Z), pescado(Y).
% (b) ¿Cómo se pregunta por las comidas con naranja sin modificar
%     el programa? comida(X, Y, naranja).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 6: Para completar un poco la información que tenemos
% sobre las comidas del restaurante vamos a mirar la lista de las
% calorías que aporta cada plato:
% * Una ración de paella aporta 200 calorías
% * Una ración de gazpacho aporta 150 calorías
% * Una ración de consomé aporta 300 calorías
% * Una ración de filete de cerdo aporta 400 calorías
% * Una ración de pollo asado aporta 280 calorías
% * Una ración de trucha aporta 160 calorías
% * Una ración de bacalao aporta 300 calorías
```

```
% * Una ración de flan aporta 200 calorías

% * Una ración de nueces con miel aporta 500 calorías

% * Una ración de naranja aporta 50 calorías

%

% Definir la relacion "calorias(X,N)" que indicará  que una

% ración de "X" tiene "N" calorías.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%


calorias(paella, 200).

calorias(gazpacho, 150).

calorias(consome, 300).

calorias(filete_de_cerdo, 300).

calorias(pollo_asado, 280).

calorias(trucha, 160).

calorias(bacalao, 300).

calorias(flan, 200).

calorias(nueces_con_miel, 500).

calorias(naranja, 50).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

% Ejercicio 7: Definir la relacion "valor_calorico(X,Y,Z,V)"

% que indicará que la comida comida(X,Y,Z) suma en total "V"

% calorías.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
```

```
valor_calorico(X, Y, Z, V):-
	comida(X, Y, Z),
	calorias(X, V1),
	calorias(Y, V2),
	calorias(Z, V3),
	V is V1+V2+V3.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% Ejercicio 8: Definir la relacion "comida_equilibrada(X,Y,Z)"
% que indicará que la comida comida(X,Y,Z) no supera las 800
% calorías.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

comida_equilibrada(X, Y, Z):-
	valor_calorico(X, Y, Z, V),
	V<800.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% Ejercicio 9:
% Considera el siguiente programa que describe algunas relaciones
% familiares

padre(andres,bernardo).
```

```
padre(andres,belen).

padre(andres,baltasar).

padre(baltasar,david).

padre(david,emilio).

padre(emilio,francisco).

madre(ana,bernardo).

madre(ana,belen).

madre(ana,baltasar).

madre(belen,carlos).

madre(belen,carmen).


% extender el programa para definir las siguientes relaciones

% familiares

% (9.1) abuelo/2

% (9.2) progenitor/2

% (9.3) nieta/2

% (9.4) antepasado/2

% (9.5) descendiente/2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%


abuelo(X, Y):-

        padre(X, Z),

        padre(Z, Y).

abuelo(X, Y):-

        padre(X, Z),
```

```
        madre(Z, Y).
progenitor(X, Y):-
        padre(X, Y).
progenitor(X, Y):-
        madre(X, Y).
mujer(belen).
mujer(ana).
mujer(carmen).
nieta(X, Y):-
        mujer(X),
        progenitor(Y, Z),
        progenitor(Z, X).
antepasado(X, Y):-
        progenitor(X, Y).
antepasado(X, Y):-
        progenitor(X, Z),
        antepasado(Z, Y).
descendiente(X, Y):-
        antepasado(Y, X).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 1:
% Consideremos el siguiente programa con información de parejas inscritas
% en el registro
```

```
libro_de_familia(
        esposo(nombre(antonio,garcia,fernandez),
            profesion(arquitecto),
                    salario(300000)),
        esposa(nombre(ana,ruiz,lopez),
            profesion(docente),
                    salario(120000)),
        domicilio(sevilla)).

libro_de_familia(
        esposo(nombre(luis,alvarez,garcia),
            profesion(arquitecto),
                    salario(400000)),
        esposa(nombre(ana,romero,soler),
            profesion(sus_labores),
                    salario(0)),
        domicilio(sevilla)).

libro_de_familia(
        esposo(nombre(bernardo,bueno,martinez),
            profesion(docente),
                    salario(120000)),
        esposa(nombre(laura,rodriguez,millan),
            profesion(medico),
                    salario(250000)),
        domicilio(cuenca)).
```

```
libro_de_familia(
        esposo(nombre(miguel,gonzalez,ruiz),
            profesion(empresario),
                salario(400000)),
        esposa(nombre(belen,salguero,cuevas),
            profesion(sus_labores),
                salario(0)),
        domicilio(dos_hermanas)).

%  (1) Definir el predicado profesion(X) que se verifique si X es una
%     profesión que aparece en el programa
%  (2) Definir el predicado primer_apellido(X) que se verifique si X es el
%     primer apellido de alguien
%  (3) Determinar el nombre completo de todas las personas que viven en Sevilla
%  (4) Definir el predicado ingresos_familiares(N), de forma que si N es
%     una cantidad que determina los ingresos totales de una familia.
%  (5) Definir el predicado pareja(Hombre,Mujer) que devuelva los nombres
%     de pila de las parejas existentes.
%  (6) Definir el predicado sueldo(X,Y) que se verifique si el sueldo de
%     la persona de nobre completo X es Y.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

profesion(X):-
```

```
        libro_de_familia(esposo(_,profesion(X),_),_,_).
profesion(X):-
        libro_de_familia(_,esposa(_, profesion(X),_),_).
primer_apellido(X):-
        libro_de_familia(esposo(nombre(_,X,_),_,_),_,_).
primer_apellido(X):-
        libro_de_familia(_,esposa(nombre(_,X,_),_,_),_).
sevillano(X):-
        libro_de_familia(esposo(X,_,_),_,domicilio(sevilla)).
sevillano(X):-
        libro_de_familia(_,esposa(X,_,_),domicilio(sevilla)).
ingresos_familiares(X):-
        libro_de_familia(esposo(_,_,salario(N1)),esposa(_,_,salario(N2)),_),
        X is N1+N2.
pareja(X, Y):-
        libro_de_familia(esposo(nombre(X,_,_),_,_),esposa(nombre(Y,_,_),_,_),_).
sueldo(X, Y):-
        libro_de_familia(esposo(X,_,salario(Y)),_,_).
sueldo(X, Y):-
        libro_de_familia(_,esposa(X,_,salario(Y)),_).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 2:
% En este ejercicio vamos a caracterizar los movimientos de las piezas del
% ajedrez. Se pide definir el predicado
```

```
%	movimiento(Pieza,Casilla_origen,Casilla_llegada)

% donde las piezas son torre, caballo, alfil, rey, dama y peón y las

% casillas las denominamos por el término e(X,Y) con X e Y entre 1 y 8.

% (Consideraremos siempre que la casilla de origen es una casilla legal)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%


movimiento(X,Y,Z):-

	pieza1(X),

	correcto(Y),

	movimiento_torre(Y,Z),

	correcto(Z).

movimiento(X,Y,Z):-

	pieza2(X),

	correcto(Y),

	movimiento_caballo(Y,Z),

	correcto(Z).

movimiento(X,Y,Z):-

	pieza3(X),

	correcto(Y),

	movimiento_alfil(Y,Z),

	correcto(Z).

movimiento(X,Y,Z):-

	pieza4(X),

	correcto(Y),

	movimiento_rey(Y,Z),
```

```
        correcto(Z).
movimiento(X,Y,Z):-
        pieza5(X),
        correcto(Y),
        movimiento_dama(Y,Z),
        correcto(Z).
movimiento(X,Y,Z):-
        pieza6(X),
        correcto(Y),
        movimiento_peon(Y,Z),
        correcto(Z).
pieza1(torre).
pieza2(caballo).
pieza3(alfil).
pieza4(rey).
pieza5(dama).
pieza6(peon).
correcto((X,Y)):-
        between(1,8,X),
        between(1,8,Y).
ok(X,_):-
        X =\= 0.
ok(_,X):-
        X =\= 0.
movimiento_torre((X1,Y1),(X2,Y2)):-
        between(-7,7,Z),
```

```
        ok(Z,0),
        X2 is X1+Z,
        Y2 is Y1.
movimiento_torre((X1,Y1),(X2,Y2)):-
        between(-7,7,Z),
        ok(0,Z),
        X2 is X1,
        Y2 is Y1+Z.
movimiento_caballo((X1,Y1),(X2,Y2)):-
        X2 is X1+1,Y2 is Y1+2;
        X2 is X1+1,Y2 is Y1-2;
        X2 is X1-1,Y2 is Y1+2;
        X2 is X1-1,Y2 is Y1-2;
        X2 is X1+2,Y2 is Y1+1;
        X2 is X1+2,Y2 is Y1-1;
        X2 is X1-2,Y2 is Y1+1;
        X2 is X1-2,Y2 is Y1-1.
movimiento_alfil((X1,Y1),(X2,Y2)):-
        between(-7,7,Z),
        ok(Z,0),
        X2 is X1+Z,
        Y2 is Y1+Z.
movimiento_rey((X1,Y1),(X2,Y2)):-
        between(-1,1,Z),
        between(-1,1,T),
        ok(Z,T),
```

```
        X2 is X1+Z,

        Y2 is Y1+T.

movimiento_dama(X,Y):-

        movimiento_torre(X,Y).

movimiento_dama(X,Y):-

        movimiento_alfil(X,Y).

movimiento_peon((X1,Y1),(X2,Y2)):-

        X2 is X1,

        Y2 is Y1+1.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Ejercicio 3: [Ex. Feb 2001]

% Usaremos la siguiente notación para representar los números enteros

%   (a) 0 es un número entero

%   (b) Si X es un número entero entonces s(X) es un número entero.

%       (s(X) representará el sucesor de X).

%   (c) Si X es un número entero entonces p(X) es un número entero.

%       (s(X) representará el predecesor de X).

% De esta manera s(s(0)) representará al 2 y p(p(p(0))) representará al -3.

% Pero p(s(s(p(0)))) también represeanta al 0 y s(s(p(s(0)))) también

% representa al 2. Diremos que una representación es minimal si en ella no

% ocurren los símbolos "p" y "s" simultáneamente.

% Definir el predicado

%                 simplifica(Rep,Rep_minimal)
```

```
% que tome como dato de entrada un número entero con la representación Rep

% y devuelva el mismo número entero con su representación minimal.

%    Ejemplo:

%

%   ?- simplifica(s(s(p(s(s(s(p(0))))))),M).

%      M = s(s(s(0))) ;

%      No

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%


entero(0).

entero(s(X)):-

            entero(X).

entero(p(X)):-

            entero(X).

simplifica(X,Y):-

            numero_sp(X,S,P),

            S > P,

            TS is S-P,

            simplifica_a_s(Y,TS).

simplifica(X,Y):-

            numero_sp(X,S,P),

            S < P,

            TP is P-S,

            simplifica_a_p(Y,TP).
```

```
simplifica(X,Y):-
	numero_sp(X,S,P),
	S =:= P,
	Y is 0.
numero_sp(0,0,0).
numero_sp(s(X),S,P):-
	numero_sp(X,S1,P),
	S is S1+1.
numero_sp(p(X),S,P):-
	numero_sp(X,S,P1),
	P is P1+1.
simplifica_a_s(0,0).
simplifica_a_s(X,S):-
	S > 0,
	S1 is S-1,
	simplifica_a_s(Y,S1),
	X=s(Y).
simplifica_a_p(0,0).
simplifica_a_p(X,P):-
	P > 0,
	P1 is P-1,
	simplifica_a_p(Y,P1),
	X=p(Y).
```

%%

```prolog
% Ejercicio 1: Definici\'on recursiva de lista

%

% Define un predicado "es_lista(X)" tal que tenga exito si "X"

% es una lista y no lo tenga en caso contrario.

%

% Ejemplos

%

%  ?- es_lista(f(a,b)).

%  No

%  ?- es_lista([]).

%  Yes

%  ?- es_lista([a,b]).

%  Yes

%

% Nota: Estudia el comportamiento de los  predicados predefinidos is_list/1 y

% proper_list/1 y comp\'aralos con es_lista/1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


es_lista([]).

es_lista([_|X]):-

            es_lista(X).


%es_lista([_|_]) :- !.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

% Ejercicio 2: Concatenacion de listas

% Define el predicado conc(L1,L2,L3) de forma que si L1 y L2 son

% listas, entonces L3 es la lista obtenida escribiendo los elementos

% de L2 a continuaci\'on de los elementos de L1. Por ejemplo,

% si L1 es [a,b] y L2 es [c,d], entonces L3 es [a,b,c,d].

%

% Ejemplos

%

% ?- conc([a,b],[c,d,e],L).

% L = [a, b, c, d, e] ;

%

% ?- conc([a,b],L,[a,b,c,d]).

% L = [c, d] ;

%

% ?- conc(L1,L2,[a,b]).

% L1 = []

% L2 = [a, b] ;

% L1 = [a]

% L2 = [b] ;

% L1 = [a, b]

% L2 = [] ;

%

% ?- conc(L1,[b|L2],[a,b,c]).

% L1 = [a]
```

% L2 = [c] ;

%

% ?- conc(_,[b|_],[a,b,c]).

% Yes

%

% ?- conc(_,[b,c|_],[a,b,c,d]).

% Yes

%

% ?- conc(_,[b,d|_],[a,b,c,d]).

% No

%

% ?- conc(_,[X],[b,a,c,d]).

% X = d ;

% No

%

% Nota: conc/3 corresponde con el predicado predefinido append/3

%%%
%%%%%%%%%%%%%%%%%%%

conc([],L,L).

conc([X|L1],L2,[X|L3]):-

    conc(L1,L2,L3).

%%%
%%%%%%%%%%%%%%%%%%%

% Ejercicio 3: Par consecutivo en una lista

```
% Define el predicado "par_en_lista(L1,L2)" donde L1 es una lista de dos
% elementos que aparecen de forma consecutiva en la lista L2.
% Ejemplos
%
%  ?- par_en_lista(L,[1,2,3,4]).
%  L = [1, 2] ;
%  L = [2, 3] ;
%  L = [3, 4] ;
%  No
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

par_en_lista([],[]).
par_en_lista([],[_]).
par_en_lista([X,Y],L):-
        conc(L1,_L2,L),
        conc(_L3,[X,Y],L1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 4: Define la relaci\'on rota(L1,L2)
% de forma que la lista L2 sea la lista L1 en la que el primer elemento se ha
% situado en la \'ultima posici\'on. Usa la relaci\'on "conc" ya definida.
%
% Ejemplos
%
```

```
% ?- rota([a,b,c,d],L).

% L = [b, c, d, a]

% Yes

%

% ?- rota(L,[b,c,d,a]).

% L = [a, b, c, d]

% Yes

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rota([],[]).

rota([X|L1],L2):-

        conc(L1,[X],L2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Ejercicio 5: Define el predicado

%                    inversa(L1,L2)

% de forma que si L1 es una lista, entonces L2 es la lista obtenida

% invirtiendo los elementos de L1. Por ejemplo, si L1 es [a,b,c]

% entonces L2 es [c,b,a].

%

% Ejemplos:

%

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

%    L = [c, b, a] ;
```

```
%    No
%
% Nota: inversa/2 coincide con el predicado predefinido reverse/2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inversa([],[]).
inversa([X|L1],L2):-
        inversa(L1,L3),
        conc(L3,[X],L2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 6: Define recursivamente la relacion
%                  borrado_1(X,L1,L2)
% de forma que L2 sea una lista obtenida eliminando una ocurrencia
% del elemento X en la lista L1.
%
% Ejemplos
%
% ?- borrado_1(a,[a,b,a],L).
% L = [b, a] ;
% L = [a, b] ;
% No
%
% ?- borrado_1(a,L,[1,2]).
```

```
% L = [a, 1, 2] ;
% L = [1, a, 2] ;
% L = [1, 2, a] ;
% No
%
% ?- borrado_1(X,[1,2,3],[1,3]).
% X = 2 ;
% No
%
% Nota: borrado_1/3 corresponde con el predicado predefinido select/3.
%       Comp\'aralo con delete/3.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


borrado(X,L1,L2):-
        conc(L5,L4,L1),
        conc(L3,[X],L5),
        conc(L3,L4,L2).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 7: Define la relaci\'on
%         insertado(X,L1,L2)
% de forma que L2 sea una lista obtenida insertando el elemento X
% en la lista L1. Usa la relaci\'on "borrado" ya definida.
%
```

```
% Sesi\'on

%

% ?- insertado(a,[1,2],L).

% L = [a, 1, 2] ;

% L = [1, a, 2] ;

% L = [1, 2, a] ;

% No

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%


insertado(X,L2,L1):-

        conc(L3,L4,L2),

        conc(L3,[X],L5),

        conc(L5,L4,L1).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

% Ejercicio 8: Define recursivamente la relaci\'on

%              permutacion_1(L1,L2)

% de forma que la lista L2 sea una permutaci\'on de los elementos de

% la lista L1. Usa la relaci\'on "insertado" definida anteriormente.

%

% Ejemplos

%

% ?- permutacion_1([a,b,c],L).

% L = [a, b, c] ;
```

```
% L = [b, a, c] ;

% L = [b, c, a] ;

% L = [a, c, b] ;

% L = [c, a, b] ;

% L = [c, b, a] ;

% No

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


permutacion_1([],[]).

permutacion_1([X|L1],L2):-

        permutacion_1(L1,L3),

        insertado(X,L3,L2).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Ejercicio 9: ¿Qu\'e t\'erminos hay que poner en lugar de los

% asteriscos para que el programa

%

%        permutacion_2([],*).

%        permutacion_2(L1,[X|L2]) :-

%            borrado_1(*,*,*),

%            permutacion_2(L3,*).

%

% sea una definici\'on de permutacion_2(L1,L2) de forma que la lista L2 sea

% una permutaci\'on de los elementos de la lista L1?
```

```
%
% Ejemplos
%
% ?- permutacion_2([a,b,c],L).
% L = [a, b, c] ;
% L = [a, c, b] ;
% L = [b, a, c] ;
% L = [b, c, a] ;
% L = [c, a, b] ;
% L = [c, b, a] ;
% No
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

permutacion_2([],[]).
permutacion_2(L1,[X|L2]) :-
	borrado(X,L1,L3),
	permutacion_2(L3,L2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 10: Define la relaci\'on
%	n_esimo(N,L,X)
% de forma que X sea el N-\'esimo elemento de la lista L.
%
% Ejemplos
```

```
%

% ?- n_esimo(2,[a,b,c,d],X).

% X = b

% Yes

%

% Nota: Estudia el comportamiento de los predefinidos nth0/3 y nth1/3

%       y comp\'aralos con n_esimo/3

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


longitud([],0).

longitud([_|L],N):-

        longitud(L,N1),

        N is N1+1.


n_esimo(1,[Y|_],Y).

n_esimo(N,[Y|L],X):-

        longitud([Y|L],N1),

        N < N1+1,

        N > 1,

        N2 is N-1,

        n_esimo(N2,L,X).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Ejercicio 1:
```

```
% Definir la relación

%      elementos_pares(L1,L2)

% de forma que si L1 es una lista de números, entonces L2 es la lista

% de los elementos pares de L1. Por ejemplo,

%    ?- elementos_pares([1,2,4,3,5],L).

%      L = [2,4] ;

%      No

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


elementos_pares([],[]).

elementos_pares([X|L1],[X|L2]):-

            0 =:= X mod 2,

            elementos_pares(L1,L2),!.

elementos_pares([_|L1],L2):-

            elementos_pares(L1,L2).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Ejercicio 2:

% Definir el predicado suma_hasta(N,S) que tome como entrada un n\'umero

% natural mayor o igual que uno y devuelva la suma de todos los n\'umeros

% naturales de 1 a N. Ejemplo de uso:

%

%   ?- suma_hasta(4,S).

%     S = 6 ;
```

```
%      No

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%


suma_hasta(1,1).

suma_hasta(N,S):-

            N > 1,

            N1 is N-1,

            suma_hasta(N1,S1),

            S is S1+N.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%

% Ejercicio 3: [Bratko-86, p.162] Definir el predicado simplifica(+E1,-E2)

% donde E2 es la expresión obtenida simplificando las sumas de los números que

% aparecen en E1  (que es una suma de números y átomos). Por ejemplo,

%     ?- simplifica(1+a+2.5,X).

%     X = a+3.5

%     Yes

%

%     ?- simplifica(1+a+4+2+b+c,X).

%     X = a+b+c+7

%     Yes

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%


simplifica(E1,E2):-
```

```
        Numeros is 0,
        Atomos is 0,
        separa(E1,Atomos,Numeros),
        junta(E2, Atomos,Numeros).


separa(X,X,0):-
        atom(X),!.
separa(X,0,X):-
        number(X),!.
separa(X+E,X+A,N):-
        atom(X),
        separa(E,A,N),!.
separa(X+E,A,N):-
        number(X),
        separa(E,A,N1),
        N is X+N1,!.

junta(A,A,0):-!.
junta(N,0,N):-!.
junta(A+N,A,N).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 4: Definir un predicado
%     n_derecha(+L,+N,?X)
```

```
% que se verifique si X es el elemento situado en la N-ésima posición,
% empezando a contar por la derecha, de la lista L. Por ejemplo,
%    ?- n_derecha([a,b,c,d,e],4,X).
%    X = b ;
%    No
%    ?- n_derecha([a,b,c,d,e],15,X).
%    No
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n_derecha(L,N,X):-
        append(_,[X|L2],L),
        length([X|L2],N).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Un \'arbol binario es vac\'io o consta de tres partes:
%  * Una raiz
%  * Un sub\'arbol izquierdo (que debe ser un \'arbol binario)
%  * Un sub\'arbol derecho (que debe ser un \'arbol binario)
%
% Consideraremos la siguiente representaci\'on
% * nil representa el \'arbol vac\'io
% * Si el \'arbol no es vac\'io, entonces tendr\'a la forma t(I,R,D) donde
%   R es la raiz, I el sub\'arbol izquierdo y D el sub\'arbol derecho.
%
```

```
%  EJEMPLO: El \'arbol
%                              a
%                           /    \
%                          b      c
%                               /
%                              d
%
% Lo representamos como t(t(nil,b,nil),a,t(t(nil,d,nil),c,nil))
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 5: Definir el predicado
%      arbolbinario(A)
% que se verifique ni A es un \'arbol binario. Por ejemplo,
%      ?- arbolbinario(nil).
%      Yes
%      ?- arbolbinario(t(t(nil,b,nil),a,t(t(nil,d,nil),c,nil))).
%      Yes
%      ?- arbolbinario(t(t(nil,b,nil),a,t(t(nil,d,e),c,nil))).
%      No
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


arbolbinario(nil).
arbolbinario(t(I,R,D)):-
```

```
            atom(R),
            arbolbinario(I),
            arbolbinario(D).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 6: Definir el predicado
%     en(Nodo,Arbol)
% que se verifique si Nodo es un nodo del \'arbol binario Arbol. Por
% ejemplo
%     ?- en(Nodo,t(t(nil,b,nil),a,t(t(nil,d,nil),c,nil))).
%        Nodo = a ;
%        Nodo = b ;
%        Nodo = c ;
%        Nodo = d ;
%        No
%     ?- en(b,t(t(nil,b,nil),a,t(t(nil,d,nil),c,nil))).
%        Yes
%     ?- en(f,t(t(nil,b,nil),a,t(t(nil,d,nil),c,nil))).
%        No
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

en(Nodo,t(I,Nodo,D)):-
            arbolbinario(I),
            arbolbinario(D).
```

```
en(Nodo,t(I,_,_)):-
        en(Nodo,I).
en(Nodo,t(_,_,D)):-
        en(Nodo,D).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 7: Definir el predicado
%     profundidad(Arbol,Prof)
% que se verifique si Prof es la profundidad nodo del \'arbol binario
% Arbol. Consideraremos que la profundidad del \'arbol vacio es 0 y la
% del \'arbol con un \'unico nodo es 1. Por ejemplo,
%     ?- prof(nil,Prof).
%        Prof = 0 ;
%        No
%     ?- prof(t(nil,a,nil),Prof).
%        Prof = 1 ;
%        No
%     ?- prof(t(t(nil,b,nil),a,t(t(nil,d,nil),c,nil)),Prof).
%        Prof = 3 ;
%        No
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


prof(nil,0).
prof(t(I,_,D),N):-
```

```
        prof(I,N1),
        prof(D,N2),
        maximo(N1,N2,M),
        N is M+1.

maximo(N1,N2,N1):-
        N1 > N2, !.
maximo(_,N2,N2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ejercicio 8: Definir el predicado
%     alinea(Arbol,Lista)
% que se verifique si Lista es la lista de los nodos del Arbol. Por
% ejemplo,
%     ?- alinea(nil,L).
%        L = [] ;
%        No
%     ?- alinea(t(nil,a,nil),L).
%        L = [a] ;
%        No
%     ?- alinea(t(t(nil,b,nil),a,t(t(nil,d,nil),c,nil)),L).
%        L = [b, a, d, c] ;
%        No
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
alinea(A,L):-
        bagof(_Nodo,en(_Nodo,A),L).
```

NUMERO NATURAL----------

```
natural(0).
natural(X) :- X>0, Y is X-1, natural(Y).
```

APLANAR----------

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

aplanar([],[]).
aplanar(X,[X]) :- atom(X).
aplanar([X|L],R) :- aplanar(X,R1), aplanar(L,R2), concatenar(R1,R2,R),!.
```

AGREGAR ELEMENTO----------

```
concatenar([],L,L).

concatenar([X|L1],L2,[X|L3]):- concatenar (L1,L2,L3).


longitud([],0).

longitud([X|L],Y) :- longitud(L,Y2), Y is 1+Y2.


agregar_elemento(L,X,Pos,[X|L]) :- Pos=<0.

agregar_elemento(L,Y,Pos,Z) :- longitud(L,Long), Pos>Long, concatenar(L,[Y],Z).

agregar_elemento([X|L],Y,Pos,[X|Z]) :- Pos2 is Pos-1, agregar_elemento(L,Y,Pos2,Z).
```

BORRAR----------

```
borrar_elem(_,[],[]).

borrar_elem(X,[X|L],L).

borrar_elem(X,[Y|L],[Y|R]) :- borrar_elem(X,L,R).
```

CONCATENAR----------

```
concatenar([],L,L).

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

COLA----------

```
cola([_|L],L).
```

FIBONACCI----------

```
fibonacci(0,1).

fibonacci(1,1).

fibonacci(N,V) :- N>=0, X is N-1, Y is N-2, fibonacci(X,A), fibonacci(Y,B), V is A+B.
```

FACTORIAL----------

```
factorial(0,1).
```

```
factorial(1,1).

factorial(N,V) :- N>0, X is N-1, factorial(X,V2), V is N*V2.
```

INVERSA----------

```
concatenar([],L,L).

concatenar([X|L1],L2,[X|L3]):- concatenar(L1,L2,L3).

inversa([],[]).

inversa([X|L1],L) :- inversa(L1,Resto), concatenar(Resto,[X],L).
```

LONGITUD----------

```
longitud([],0).

longitud([X|L],Y) :- longitud(L,Y2), Y is 1+Y2.
```

MAXIMO----------

```
mayor_elem([X],X).

mayor_elem([X|L],M) :- mayor_elem(L,M2), X>=M2, M=X.

mayor_elem([X|L],M) :- mayor_elem(L,M2), X<M2, M=M2.
```

MINIMO----------

```
menor_elem([X],X).

menor_elem([X|L],M) :- menor_elem(L,M2), X=<M2, M=X.

menor_elem([X|L],M) :- menor_elem(L,M2), X>M2, M=M2.
```

ORDENAR DE MAYOR A MENOR----------

```
mayor_elem([X],X).

mayor_elem([X|L],M) :- mayor_elem(L,M2), X>=M2, M=X.

mayor_elem([X|L],M) :- mayor_elem(L,M2), X<M2, M=M2.

borar_elem(_,[],[]).

borar_elem(X,[X|L],L).

borar_elem(X,[Y|L],[Y|R]) :- borar_elem(X,L,R).
```

```
ordenar([],[]).

ordenar(L,[M|O]) :- mayor_elem(L,M), borar_elem(M,L,R), ordenar(R,O).
```

ORDENAR DE MENOR A MAYOR----------

```
menor_elem([X],X).

menor_elem([X|L],M) :- menor_elem(L,M2), X=<M2, M=X.

menor_elem([X|L],M) :- menor_elem(L,M2), X>M2, M=M2.


borar_elem(_,[],[]).

borar_elem(X,[X|L],L).

borar_elem(X,[Y|L],[Y|R]) :- borar_elem(X,L,R).


ordenar([],[]).

ordenar(L,[M|O]) :- menor_elem(L,M), borar_elem(M,L,R), ordenar(R,O).
```

OPERAR----------

```
operar(X,Y,'+',R) :- R is X+Y.

operar(X,Y,'-',R) :- R is X-Y.
```

```
operar(X,Y,'*',R) :- R is X*Y.

operar(X,Y,'/',R) :- Y=\=0, R is X/Y.


operar2(X,Y,Op,R) :- (Op='/',Y=0)->R='Error matemático'; (E=..[Op,X,Y], R is E).




SUBLISTA??????????????????----------
--------------------------------------------------


lenght([],0).

lenght([P|R],X) :- lenght(R,Aux), X is (1+Aux).


valida_lista(L,P,C) :- P > 0, lenght(L,Lenght), P =< Lenght, C =< (Lenght-P+1).


unir(Pr,[],1,[Pr]).

unir(Pr,L,0,[]).

unir(Pr,[Pr2|L],C,X) :- C > 0, NovaC is (C-1), unir(Pr2,L,NovaC,Aux), concatenar([Pr],Aux,X).


sublista(L,P,0,[]).

sublista(L,P,C,X) :-  valida_lista(L,P,C), sublista_aux(L,P,C,X).


sublista_aux([Pr|R],P,C,X) :- P > 1, NovoP is (P-1), sublista_aux(R,NovoP,C,X).

sublista_aux([Pr|R],P,C,X) :- unir(Pr,R,C,X).
```

PAR O
IMPAR----------

```
longitud([],0).

longitud([X|L],Y) :- longitud(L,Y2), Y is 1+Y2.


parOimpar([],PoI) :- PoI=par.

parOimpar(L,PoI) :- longitud(L,Long), Resto is Long mod 2, Resto=:=0, PoI=par.

parOimpar(L,PoI) :- longitud(L,Long), Resto is Long mod 2, Resto=\=0, PoI=impar.
```

PALINDROMO----------

```
concatenar([],L,L).

concatenar([X|L1],L2,[X|L3]) :- concatenar(L1,L2,L3).


inversa([],[]).

inversa([X|L1],L) :- inversa(L1,Resto), concatenar(Resto,[X],L).


palindromo(L) :- inversa(L,L).
```

PERTENECE----------

pertenece(X,[X|_]).

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

PRIMERO----------

primero([X|_],X).

QUITAR
ELEMENTO----------

quitar_elemento([],_,[]).

quitar_elemento([X|L],X,Z) :- quitar_elemento(L,X,Z).

quitar_elemento([X|L],W,[X|Z]) :- quitar_elemento(L,W,Z).

SUBCONJUNTO----------

```
pertenece(X,[X|_]).

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


subconjunto([],L).

subconjunto([X|L1],L2) :- pertenece(X,L2), subconjunto(L1,L2).
```

```
%Comparacion de listas(devuelve menor)

%Importante: Las dos listas deben tener la misma cantidad de elementos, sino fallara!!!

%Argumentos: [H1|T1] es la primer lista

%            [H2|T2] es la segunda lista

%            Out la menor lista

lscmp([],[],[]):- !.

lscmp([],_,_):- !, fail.

lscmp(_,[],_):- !, fail.

lscmp([H|T1],[H|T2],Out):- !, lscmp(T1,T2,Lprev), append([H],Lprev,Out), !.

lscmp([H1|T1],[H2|T2],Out):- length(T1,S), length(T2,S), (H1<H2 -> append([H1],T1,Out);
append([H2],T2,Out)).

%Nota los lenght son para que termine si es que no son de la misma dimension
```

%Comparacion de listas(devuelver True o False)

%Importante: Las dos listas deben tener la misma cantidad de elementos, sino fallara!!!

%Argumentos: [H1|T1] es la primer lista

% [H2|T2] es la segunda lista

%lscmp([],[]):- !, fail.

%lscmp([],_):- !, fail.

%lscmp(_,[]):- !, fail.

%lscmp([H|T1],[H|T2]):- !, lscmp(T1,T2).

%lscmp([H1|T1],[H2|T2]):- !, length(T1,S), length(T2,S), H1<H2.

%Nota los lenght son para que termine si es que no son de la misma dimension

---

%Comparacion de listas lexicografica(devuelver True o False)

%Importante: No es nesesario que tengan la misma longitud

%Argumentos: [H1|T1] es la primer lista

% [H2|T2] es la segunda lista

```
lscmp([],[]):- !, fail.

lscmp([],_):- !, true.

lscmp(_,[]):- !, fail.

lscmp([H|T1],[H|T2]):- !, lscmp(T1,T2).

lscmp([H1|_],[H2|_]):- H1<H2.
```

%Comparacion de reales

```
less(A,B):- (rational(A);float(A)), (rational(B);float(B)), A<B, !.
```

%Comparacion de listas(elegir el lscmp que se quiera comparacion lexicografica o entre listas de sub-listas de igual longitud)

```
less(A,B):- compound(A), compound(B), lscmp(A,B), !.
```

%Operador gnerico de comparacion <

```
:-op(700,xfx,less).
```

```
equal(A,A):- !.


%Operador gnerico de asignacion =

:-op(700,xfx,equal).
```

```
%Minimo Generico para listas de escalares y de sublistas(por comparacion lexicografica o de igual longitud)

%Argumentos: [H|T] es la lista de la que se sacara el minimo

%            Out es el minimo de la lista L

min([A],A):- !.

min([H|T],Out):- min(T,Mprev), (Mprev less H -> Out equal Mprev; Out equal H).


%Maximo Generico para listas de escalares y de sublistas(por comparacion lexicografica o de igual longitud)

%Argumentos: [H|T] es la lista de la que se sacara el minimo

%            Out es el minimo de la lista L

max([A],A):- !.

max([H|T],Out):- max(T,Mprev), ( H less Mprev -> Out equal Mprev; Out equal H).
```

%Borrado Generico(como habeces hay drama con el delete)

%Argumentos: A es el elemento a borrar de la lista

%	[H|T] es la lista de donde borrar A

%	Out la lista sin A

```
erase(_,[],[]):- !.

erase(A,[A|T],T):- !.

erase(A,[H|T],Out):- erase(A,T,Dprev), append([H],Dprev,Out).
```

%Ordenacion por Seleccion Generico(menor a mayor <-> less ; mayor-igual a menor <-> greaterequal)

%Argumentos: L es la lista a ordenar

%	Out la lista L ordenada

```
s_sort([],[],less):- !.

s_sort(L,Out,less):- !, min(L,Val), erase(Val,L,Lnew), s_sort(Lnew,Lsort,less), append([Val],Lsort,Out).
```

```
s_sort([],[],greaterequal):- !.

s_sort(L,Out,greaterequal):- !, max(L,Val), erase(Val,L,Lnew), s_sort(Lnew,Lsort,greaterequal), append([Val],Lsort,Out).



%Find Generico(busca tanto en listas de escalares como en lisats de sub-listas de dimension genericas)

%Argumentos: A es el elemento a buscar

%            [H|T] es la lista a ordenar

find(A,[A|_]):- !, true.

find(A,[_|T]):- find(A,T).



%Predicado, pregunta si vacio Generico

empty([]).



%Factorial

%Argumentos: N el numero del que se quiere el factorial

%            Out  es el resultado del factorial

fact(0,1):- !.

fact(N,_):- (N<0 -> !, fail; fail).

fact(N,Out):- Nprev is N-1, fact(Nprev,Fprev), Out is Fprev*N.
```

%Combinatoria

%Argumentos: N es el numero de tipos de elementos

% K es el numero de longitud de combinacion

% Out es el resultado de la combinacion

```
c(N,N,1):- !.

c(_,0,1):- !.

c(N,K,_):- (K>N -> !, fail; fail).

c(N,K,Out):- Nprev is N-1, Kprev is K-1, c(Nprev,Kprev,C1), c(Nprev,K,C2), Out is (C1+C2).
```

%Permutacion por division de factoriales

%Argumentos: N es el numero de tipos de elementos

% K es el numero de longitud de permutacion

% Out es el resultado de la permutacion

```
p(N,N,Out):- fact(N,Out), !.

p(_,K,_):- (K<0 -> !,fail;fail).

p(N,K,Out):- fact(N,P1), Sub is N-K, fact(Sub,P2), Out is P1/P2.
```

%Devuelve cualquier elemento de la lista [H|T]

```
get_some([H|_],Out):- Out is H.
```

```
get_some([_|T],Out):- get_some(T,Out).


%Tira todas las permutaciones de la lista L

all_permutation([],[]).

all_permutation(L,S):- get_some(L,Some), erase(Some,L,L2), all_permutation(L2,Lprev),
append([Some],Lprev,S).


%Tira todas las permutaciones de K elementos de la lista L

all_permutation(0,[],[]):- !.

all_permutation(0,_,[]):- !.

all_permutation(K,L,S):- Kprev is K-1, get_some(L,Some), erase(Some,L,L2),
all_permutation(Kprev,L2,Lprev), length(Lprev,Size), Size=:=Kprev, append([Some],Lprev,S).


%Tira todas las combinaciones de K elementos de la lista [H|T]

all_combination(0,[],[]):- !.

all_combination(0,_,[]):- !.

all_combination(K,[H|T],Out):- Kprev is K-1, all_combination(Kprev,T,Cprev), length(Cprev,Size),
Size=:=Kprev, append([H],Cprev,Out).

all_combination(K,[_|T],Out):- all_combination(K,T,Out).
```

```
%Aplana lalista osea:
%                           [1,[2,3,[4,5]],[7]] => aplana => (1,2,3,4,5,7)
%Argumentos: [H|T] es la lista a aplanar
%            Out  es la lista aplanada
aplanar([],[]):- !.
aplanar([H|T],Out):- compound(H), !, aplanar(H,Lsub), aplanar(T,Lnext), append(Lsub,Lnext,Out).
aplanar([H|T],Out):- aplanar(T,Lnext), append([H],Lnext,Out).
```

```
%Recibe una lista L y sale por R toda combinacion de los elementos de la lista L
list_comb([],[]).
list_comb([E|L],R):- list_comb(L,Raux), append([E],Raux,R).
list_comb([_|L],R):- list_comb(L,R).

%Recibe una lista L y sale por R la sumaoria de los elementos de la lista
sumatoria([],0).
sumatoria([E|L],R):- sumatoria(L,R2), R is (E+R2).
```

```
%Recibe L y X, la lista de elementos y lo que debe sumar la suma de la combiancion
%Y sale por R las combinaciones de elementos de L que sumados dan X
probar(L,X,R):- list_comb(L,CombL), sumatoria(CombL,X), R = CombL.
```

```
perm(Lista,[H|Perm]) :- borrar(H,Lista,Rest),perm(Rest,Perm).
perm([],[]).

borrar(X,[X|T],T).
borrar(X,[H|T],[H|NT]) :- borrar(X,T,NT).

comb(0,_,[]).
comb(N,[X|T],[X|Comb]) :- N>0,N1 is N-1,comb(N1,T,Comb).
comb(N,[_|T],Comb) :- N>0,comb(N,T,Comb).

comb2(_,[]).
comb2([X|T],[X|Comb]) :- comb2(T,Comb).
comb2([_|T],[X|Comb]) :- comb2(T,[X|Comb]).
```

```
comb_rep(0,_,[]).

comb_rep(N,[X|T],[X|RComb]) :- N>0,N1 is N-1,comb_rep(N1,[X|T],RComb).

comb_rep(N,[_|T],RComb) :- N>0,comb_rep(N,T,RComb).


varia(0,_,[]).

varia(N,L,[H|Varia]) :- N>0,N1 is N -1, borrar(H,L,Rest),varia(N1,Rest,Varia).


varia_rep(0,_,[]).

varia_rep(N,L,[H|RVaria]) :- N>0,N1 is N-1,borrar(H,L,_),varia_rep(N1,L,RVaria).
```

PARTE UNA LISTA EN UN DETERMINADO PUNTO

```
forma1ra([X,Y|_], [X]):- Y='=', !.

forma1ra([X|C],L):- forma1ra(C, L1), append([X], L1, L).
```

```
forma2da([Y|C], C):- Y='=', !.

forma2da([_|C], L):- forma2da(C, L).
```

```
pertenece(X,[X|_]).

pertenece(X,[Y|YC]):- pertenece(X,YC).


%-------------------------------------------------------------------------


conc([],J,J).

conc([X|X1],J,[X|L]):- conc(X1,J,L).


%-------------------------------------------------------------------------


inv([],[]).

inv([X|Y],L):- inv(Y,L1), conc(L1,[X],L).


%-------------------------------------------------------------------------


long([_],1):- !.

long([_|Y], S):- long(Y,S1), S is S1+1.


%-------------------------------------------------------------------------


cpos([],0).

cpos([K|X],C):- K@>=0 -> cpos(X,C1) -> C is C1+1,!.
```

```
cpos([K|X],C):-  cpos(X,C1) -> C is C1.

%-------------------------------------------------------------------------

sep([],[],[]).

sep([X|Y],P,N):- X@>=0 -> sep(Y,P1,N),conc([X],P1,P),!.

sep([X|Y],P,N):- sep(Y,P,N1),conc([X],N1,N).

%61646D696E

%-------------------------------------------------------------------------

ma([X], X) :- !.

ma([X1, X2 | L], X) :- X1 @>= X2 -> ma([X1|L], X); ma([X2|L], X).

mi([X], X) :- !.

mi([X1, X2 | L], X) :- X1 @=< X2 -> mi([X1|L], X); mi([X2|L], X).

mm([],0,0).

mm(X,MA,MI):- ma(X,MA), mi(X,MI).

%-------------------------------------------------------------------------

f(X, 0) :- X < 3,!.

f(X, 2) :- X >= 3, X < 6,!.

f(X, 4) :- X >= 6,!.
```

```
%-----------------------------------------------------------------------------

conectado(A,B):- A=a,B=b,!.

conectado(A,D):- A=a,D=d,!.

conectado(B,C):- C=c,B=b,!.

conectado(C,G):- G=g,C=c,!.

conectado(D,F):- D=d,F=f,!.

conectado(E,F):- E=e,F=f,!.

conectado(F,G):- F=f,G=g,!.

conectado(G,S):- G=g,S=salida.


salid(S1,[]):-  S1=salida,!.

salid(X,[Y|C]):- conectado(X,Y) , salid(Y,C).


%-----------------------------------------------------------------------------


pertenec(X,[X|_]):- !.

pertenec(X,[_|C]):-  pertenec(X,C).


%-----------------------------------------------------------------------------


snum([],[]).

snum([X|XC],Y):- number(X), snum(XC,Y1), conc([X],Y1,Y), !; snum(XC,Y1), conc([],Y1,Y).


%-----------------------------------------------------------------------------
```

```
dic([sanar, hola, sabana, sabalo, prueba, computadora, cartera, mate, termo, mesa, silla, sarna]).


semejanza([],[],0):- !.

semejanza([],[Y|Y1],S):- semejanza([],Y1,S1), S is S1-1.

semejanza([X|X1],[],S):- semejanza([],X1,S1), S is S1-1,!.

semejanza([X|X1],[Y|Y1],S):- X==Y, semejanza(X1,Y1,S1), X==Y -> S is S1+1; semejanza(X1,Y1,S1), S
is S1-1.


simil(X,[],[]):- !.

simil(X,[Y|YC], L):- atom_chars(X, X2), atom_chars(Y, Y2), semejanza(X2,Y2,S), S@>0 ->
simil(X,YC,L1), conc([Y],[S],AUX),conc(AUX,L1,L); simil(X,YC,L).


busco(X,L):- dic(Y), pertenece(X,Y) -> conc([X],[],L), !; dic(Y), simil(X,Y,L).


%----------------------------------------------------------------------------


reem(_,_,_,C,_,_):- C@<0, write('4to parametro invalido'),!.

reem(_,_,_,_,C2,_):- C2@<(-1), write('5to parametro invalido'),!.

reem(L,X,Y,C,V,LF):- re(L,X,Y,C,V,LF).

re([],_,_,_,_,[]):- !.

re(L,_,_,_,0,L):- !.

re(L,X,Y,1,-1,LF):- retodo(L,X,Y,LF),!.

re(L,X,Y,1,V,LF):- repa(L,X,Y,V,LF),!.

re(L,X,Y,C,V,LF):- refin(L,X,Y,C,V,LF),!.


retodo([],_,_,[]):- !.
```

```
retodo([L|LC],X,Y,LF):- L=X -> retodo(LC,X,Y,LF1), conc([Y],LF1,LF); retodo(LC,X,Y,LF1),
conc([L],LF1,LF).


repa([],_,_,_,[]):- !.

repa(L,X,Y,0,L):- !.

repa([L|LC],X,Y,V,LF):- L=X, V1 is V-1, repa(LC,X,Y,V1,LF1), conc([Y],LF1,LF);  repa(LC,X,Y,V,LF1),
conc([L],LF1,LF).


refin([],_,_,_,_,[]):- !.

refin(L,X,Y,1,V,LF):- repa(L,X,Y,V,LF),!.

refin([L|LC],X,Y,C,V,[L|LF]):-  C1 is C-1, refin(LC,X,Y,C1,V,LF).


%---------------------------------------------------------------------------


es_conjunto([]):- !.

es_conjunto([L|Lc]):- not(pertenec(L,Lc)), es_conjunto(Lc).


pertenece_conj([],_):- !.

pertenece_conj(X,C):- pertenec(X,C).


agregar_conj(X,C,Cf):- pertenec(X,C)-> conc(C,[],Cf),!; conc(C,[X],Cf).


unir_conj([],C2,C2):- !.

unir_conj([X|C1],C2,C):- unir_conj(C1,C2,Ca), agregar_conj(X,Ca,C).


inter_conj([],C2,[]):- !.
```

```
inter_conj([X|C1],C2,C):- pertenece_conj(X,C2) -> inter_conj(C1,C2,Cf), conc([X],Cf,C),!;
inter_conj(C1,C2,C).


dif_conj([],C2,[]):- !.

dif_conj([X|C1],C2,C):- not(pertenece_conj(X,C2)) -> dif_conj(C1,C2,Cf), conc([X],Cf,C),!;
dif_conj(C1,C2,C).


crea_conj([X],[X]):- !.

crea_conj([X|L],C):- crea_conj(L,C1), not(pertenece_conj(X,C1)), conc([X],C1,C),!; crea_conj(L,C).


%----------------------------------------------------------------------------


con(A,B):- A=a , B=b,!; A=b, B=a,!.

con(B,C):- B=b , C=c,!; B=c, C=b,!.

con(C,D):- C=c , D=d,!; C=d, D=c,!.

con(E,D):- E=e , D=d,!; E=d, D=e,!.

con(E,H):- E=e , H=h,!; E=h, H=e,!.

con(E,G):- E=e , G=g,!; E=g, G=e,!.

con(E,C):- E=e , C=c,!; C=e, E=c,!.

con(G,H):- G=g , H=h,!; G=h, H=g,!.

con(F,G):- F=f , G=g,!; F=g, G=f,!.

con(A,C):- A=a , C=c,!; C=a, A=c,!.

con(A,F):- A=a , F=f,!; A=f, F=a,!.

con(C,F):- C=c , F=f,!; C=f, F=c,!.

con(C,G):- C=c , G=g,!; C=g, G=c,!.
```

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

elimino([X|_],[X|L],L):- !.

elimino([X|_],[Y|L],L1):- elimino([X],L,L2),conc([Y],L2,L1).


aux(A,[],[]):- !.

aux(A,[X|L1],C):- con(A,X) -> conc([X],[],C),!; aux(A,L1,C).


ciclo([A|_],[],[]):- !.

ciclo([A|_],[X|L], C):- aux(A,[X|L],Y), elimino(Y,[X|L],Z), ciclo(Y,Z,Y1), conc(Y,Y1,C),!.


%ciclo([a],[b,c,d,e,f,h,g],L).

%------------------------------------------------------------------------------


reinas([[1,1],[3,4],[5,7],[8,8],[6,2]]).


diagonal_1(X,_,_,[]):- X@>8, !.

diagonal_1(_,Y,_,[]):- Y@>8, !.

diagonal_1(X,Y,R,L):- X1 is (X+1),
                      Y1 is (Y+1),
                      pertenec([X,Y],R),
                      diagonal_1(X1,Y1,R,L1),
                      conc([[X,Y]],L1,L),!;
                      X1 is (X+1),
                      Y1 is (Y+1),
                      diagonal_1(X1,Y1,R,L).
```

```
diagonal_2(X,_,_,[]):- X@<0, !.

diagonal_2(_,Y,_,[]):- Y@<0, !.

diagonal_2(X,Y,R,L):- X1 is (X-1),
                      Y1 is (Y-1),
                      pertenec([X,Y],R),
                      diagonal_2(X1,Y1,R,L1),
                      conc([[X,Y]],L1,L),!;
                      X1 is (X-1),
                      Y1 is (Y-1),
                      diagonal_2(X1,Y1,R,L).


diagonal_3(X,_,_,[]):- X@<0, !.

diagonal_3(_,Y,_,[]):- Y@>8, !.

diagonal_3(X,Y,R,L):- X1 is (X-1),
                      Y1 is (Y+1),
                      pertenec([X,Y],R),
                      diagonal_3(X1,Y1,R,L1),
                      conc([[X,Y]],L1,L),!;
                      X1 is (X-1),
                      Y1 is (Y+1),
                      diagonal_3(X1,Y1,R,L).


diagonal_4(X,_,_,[]):- X@>8, !.

diagonal_4(_,Y,_,[]):- Y@<0, !.

diagonal_4(X,Y,R,L):- X1 is (X+1),
                      Y1 is (Y-1),
```

```
                pertenec([X,Y],R),
                diagonal_4(X1,Y1,R,L1),
                conc([[X,Y]],L1,L),!;
                X1 is (X+1),
                Y1 is (Y-1),
                diagonal_4(X1,Y1,R,L).

vertical(X,_,_,[]):- X@>8 ,!.
vertical(X,Y,R,L):-    X1 is X+1,
                pertenec([X,Y],R),
                vertical(X1,Y,R,L1),
                conc([[X,Y]],L1,L),!;
                X1 is (X+1),
                vertical(X1,Y,R,L).

lateral(_,Y,_,[]):- Y@>8 ,!.
lateral(X,Y,R,L):-   Y1 is Y+1,
                pertenec([X,Y],R),
                lateral(X,Y1,R,L1),
                conc([[X,Y]],L1,L),!;
                Y1 is (Y+1),
                lateral(X,Y1,R,L).

validar(X,_,_):-  X@>8 -> write('coordena invalida'), !;
            X@<0-> write('coordena invalida'),!.
validar(_,Y,_):-  Y@>8 -> write('coordena invalida'), !;
```

```
        Y@<0-> write('coordena invalida'),!.
validar(X,Y,_):- reinas(R), pertenec([X,Y],R) -> write('coordenada ocupada'),!.
validar(X,Y,L):- reinas(R),
        diagonal_1(X,Y,R,L1),
        diagonal_2(X,Y,R,L2),
        diagonal_3(X,Y,R,L3),
        diagonal_4(X,Y,R,L4),
        vertical(1,Y,R,L5),
        lateral(X,1,R,L6),
        conc(L1,L2,LL1),
        conc(L3,L4,LL2),
        conc(L5,L6,LL3),
        conc(LL1,LL2,LLL1),
        conc(LL3,LLL1,L).

%--------------------------------------------------------------------------------
%cuenta seguidos iwales

cue_iw(X,[],0):- !.
cue_iw(X,[X|L],C):- cue_iw(X,L,C1), C is C1+1,!.
cue_iw(X,[Y|L],0).


cuenta_iw([X],1):- !.
cuenta_iw([X|L], C):- cue_iw(X,[X|L],C).
```

```
%-----------------------------------------------------------------------------

%Corta n elementos


corta([],_,[]):- !.

corta(L,0,L):- !.

corta([X|L],N,Lf):- N1 is N-1, corta(L,N1,Lf).


%-----------------------------------------------------------------------------

%lista con cant de elem repetidos


repet([],[]):- !.

repet(L,Lf):- cuenta_iw(L,C), corta(L,C,LL), repet(LL, L1), conc([C],L1,Lf).


%-----------------------------------------------------------------------------


fibonacci(0,0):- !.

fibonacci(1,1):- !.

fibonacci(N,C):- N1 is N-1,N2 is N-2,fibonacci(N1,C1), fibonacci(N2,C2), C is C1+C2.


%-----------------------------------------------------------------------------


agregar_elem([],X,_,[X]):- !.

agregar_elem(L,X,C,[X|L]):- C@=<1, !.

agregar_elem([Y|L],X,C,Lf):- C1 is C-1, agregar_elem(L,X,C1,La), conc([Y],La,Lf).


%-----------------------------------------------------------------------------
```

```
parimpar([],par):- !.

parimpar(L,X):- long(L,S), R is S mod 2, R=0 -> X=par; X=impar.

%-------------------------------------------------------------------------

par(X):- X1 is X mod 2, X1=0.

%-------------------------------------------------------------------------

sub(_,0,[]):- !.

sub([X|L],C,SL):- C1 is C-1, sub(L,C1,S), conc([X],S,SL).

sublista([],_,_,[]):- !.

sublista([X|L],X,C,SL):- sub([X|L],C,SL),!.

sublista([X|L],Y,C,SL):- sublista(L,Y,C,SL).

%-------------------------------------------------------------------------

cuentaconsec([X],1):- !.

cuentaconsec([X,X|L],C):- cuentaconsec([X|L],C1) , C is C1+1, !.

cuentaconsec([X,Y|L],1).

%-------------------------------------------------------------------------

%ejercicio scheme (parcial)
```

```
zipsep([],[]):- !.

zipsep([X|L1],L):- cuentaconsec([X|L1],C), corta([X|L1],C,LL), zipsep(LL,LLL), conc([[X, C]], LLL, L).


%---------------------------------------------------------------------------

%ejercicio scheme (parcial)


formarlista(0,_,[]):- !.

formarlista(_,0,[]):- !.

formarlista(X,Y,L):- X1 is X//2, Y1 is Y*2, formarlista(X1, Y1, LL), conc([[X,Y]],LL,L).


suma(0,[]):-!.

suma(Z,[[X|Y]|C]):- par(X) -> suma(Z,C); suma(Z1,C), Z is Y+Z1.


sumarusa(X,Y,Z):- formarlista(X,Y,L), suma(Z,L).


%---------------------------------------------------------------------------


conca(_,L2,0,_,L1):- !.

conca(L1,_,_,0,L2):- !.

conca(_,_,0,0,[]):- !.

conca([],_,_,_,[]):- !.

conca(_,[],_,_,[]):- !.

conca([La|L1],[Lb|L2],X,Y,L):- sublista([La|L1],La,X,LL1),
                               sublista([Lb|L2],Lb,Y,LL2),
                               corta([La|L1],X,LL3),
                               corta([Lb|L2],Y,LL4),
```

```
                    conca(LL3,LL4,X,Y,LLL),
                    conc(LL1,LL2,Aux),
                    conc(Aux,LLL,L).

%-------------------------------------------------------------------------
%preparcial 2009
%ejercicio 6

operar(X,Y,-,R):- R is X - Y,!.
operar(X,Y,+,R):- R is X + Y,!.
operar(X,Y,*,R):- R is X * Y,!.
operar(X,Y,/,R):- Y=0 -> display('division por cero'); R is X / Y,!.

%-------------------------------------------------------------------------
%ejercicio 5
/*
aux(X, Y1, [ X | Y1 ] ).
aux(X,[ Y | C ],[ Y | D ]):- aux(X, C, D).

permu([],[]).
permu([ X | Y1 ], L ):- permu(Y1, L1), aux(X, L1, L).

es_suma([X],X,X,[X]):- !.
es_suma([X],X,Y,[X]):- Y@>X, !.
es_suma([],0,_,[]):- !.
```

```
es_suma([X|L],Y,C,LL):- es_suma(L,Y1,C,LLL), Y is X+Y1, Y@<C ->  conc([X],LLL,LL),!; Y@=C -> conc([X],LLL,LL),true.


probar([],_,[]).

probar(L,X,Lf):- permu(L,LP), probar(LP,X,LLf), es_suma(LP,C,X,LS) -> conc(LS,LLf,Lf); permu(L,LP), probar(LP,X,LLf).

*/

%------------------------------------------------------------------------------


probar([], 0, []).

probar([X|C], N, L):- N>=X, N1 is N-X, probar(C, N1, L1), conc([X], L1, L).

probar([X|C], N, L):- probar(C, N, L).


%------------------------------------------------------------------------------


%camino(a, c, Camino, Penalidad).

%Camino = [[a, b], [b, c]]

%Penalidad = 12


aristas([a,b,c,d,e]).


arista(a, b, 5).

arista(b, c, 7).

arista(c, d, 3).

arista(a, c, 2).

arista(d, e, 3).
```

```
arista(e, a, 4).


unida(X,Y,P):- arista(X,Y,P1), P is P1,!; arista(Y,X,P1), P is P1.


%camino(X,Y,C,P):- unida(X,Y1,P1), Y1=Y, P is P1, conc([[X,Y]],[],C),!.

%camino(X,Y,C,P):- unida(X,Y1,P1), camino(Y1,Y,C1,P2), P is P1+P2, conc([[Y1,Y]],C1,C).


%--------------------------------------------------------------------------

%pre-parcial 2010

%ejercicio 7


pert(X, []):- fail,!.

pert(X,[X|C]):- !.

pert(X,[_|C]):- pert(X,C).


quitar([],[]):- !.

quitar([X|C],L):- not(pert(X,C)), quitar(C,L1), conc([X],L1,L),!.

quitar([X|C],L):- quitar(C,L).


%--------------------------------------------------------------------------

%ejercicio 8}


sumar([],0,0):- !.

sumar([X],X,0):- !.

sumar([X,Y|C],SI,SP):- sumar(C,SI1,SP2), SI is SI1+X, SP is SP2+Y.
```

```
%--------------------------------------------------------------------------

%ejercicio 4

%[[a, [b]], [b, [c, d]], [c, [e]], [d, [a, e]], [e, [a]]]

%[[a, b, c, d, e], [[a, b], [b, c], [b, d], [c, e], [d, a], [d, e], [e, a]]]


obtener_nodos([],[]):- !.

obtener_nodos([[X|_]|C],L):- obtener_nodos(C,L1), conc([X],L1,L).


apl([], []):- !.

apl([X|L], L2) :- X = [_|_], !, apl(X, X1), apl(L, L1), append(X1, L1, L2),!.

apl([X|L], [X|L1]) :-  apl(L, L1).


aristas_aux([X|[]],[]):- !.

aristas_aux([X,Y|C],A):- aristas_aux([X|C],A1), conc([[X,Y]],A1,A).


obtener_aristas([],[]):- !.

obtener_aristas([X|R],A):-  obtener_aristas(R,A1), apl(X,LL), aristas_aux(LL,L), conc(L,A1,A).


chupamelaa(L1,Lf):- obtener_nodos(L1,LN), obtener_aristas(L1,LA), conc([LN],[LA],Lf).


%fin pre-parcial eaeaeaeae

%--------------------------------------------------------------------------


conc([],J,J).

conc([X|X1],J,[X|L]):- conc(X1,J,L).
```

```
invertir([],[]):- !.

invertir([X|C],L):- invertir(C,L1), conc(L1,[X],L).


invierte([],[]):- !.

invierte([X|L],LL):- atomic(X), invierte(L,P), conc(P,[X],LL),!.

invierte([X|L],I):- invierte(X,V), invierte(L,I1), conc(I1,[V],I).
```

```
% BIBLIOTECA PROLOG - Versión 2.01 - 05/2005 Autores: RB, JJC, MNG, CEP

%

% CODIGO PARA OPERACIONES CON LISTAS

% GENERALIDADES

% Determinar si lo que se recibe es una lista

% Ejemplo: lista(1). No

% lista([1,2]). Yes

lista([]):-!.

lista([_X|Y]):-lista(Y).

% Devuelve la longitud de la lista que se le pasa por argumento.

% Ejemplo: long([1,3,7,4],X). X=4

long([],0):-!.

long([_X|Y],S):-long(Y,T), S is T+1.

% Maximo elemento de una lista

% Ejemplo: maximo([1,5,3,-2],X). X=5

maximo([X],X).

maximo([X|Xs],X):- maximo(Xs,Y), X >=Y.
```

```
maximo([X|Xs],N):- maximo(Xs,N), N > X.

% Mínimo elemento de una lista

% Ejemplo: minimo([1,5,3,-2],X). X=-2

minimo([X],X).

minimo([X|Xs],X):- minimo(Xs,Y), X =< Y.

minimo([X|Xs],N):- minimo(Xs,N), N < X.

% Determina si una lista es estrictamente creciente

% Ejemplo: crece([1,2,3]). Yes

% crece([2,2,3]). No

crece([_X]).

crece([X,Y|Z]):-X<Y,crece([Y|Z]).

% Determina si una lista es estrictamente decreciente

% Ejemplo: decrece([3,2,1]). Yes

% decrece([3,2,2]). No

decrece([_X]).

decrece([X,Y|Z]):-X>Y,decrece([Y|Z]).

% Determina si dos listas son iguales en todos los niveles

% Ejemplo: listaIgual([1,2,[5,2],3],[[1,2,[5,2],3],9]). No

% listaIgual([1,2,[5,2],3],[1,2,[5,2],3]). Yes

listaIgual([],[]):-!.

listaIgual([X|M],[X|R]):-listaIgual(M,R).

listaIgual([X|M],[T|R]):-lista(X),lista(T),listaIgual(X,T),listaIgual(M,R).

% Determina si una lista es capicua

% Ejemplo: capicua([n,e,u,q,u,e,n]). Yes

% capicua([n,e,u,q,[u,e],n]). No

% capicua([n,e,[u,q,u],e,n]). Yes
```

```
capicua([]):-!.

capicua(L):-invertir(L,R),listaIgual(L,R).

% Determina si la primer lista es prefijo de la segunda

% Ejemplo: prefijo([1,2,3],[1,2,3,4,5]). Yes

% prefijo([1,5,3],[1,2,3,4,5]). No

prefijo([],_M):-!.

prefijo([_X],[_X|_M]):-!.

prefijo([_X|L],[_X|M]):-prefijo(L,M).
```

2

```
% Determina si la primer lista es posfijo de la segunda.

% Ejemplo: posfijo([3,5,4],[1,2,6,5,7,8,3,5,4]). Yes

% posfijo([3,5,4],[1,2,6,5,7,8,3,6,4]). No

posfijo(L,L1):-invertir(L,X),invertir(L1,Y),prefijo(X,Y).

% Determina si la primer lista es sublista de la segunda

% Ejemplo: subLista([3,2],[1,6,2,3]). No

% subLista([3,2],[1,6,3,2]). Yes

subLista([],_L):-!.

subLista(L,[X|M]):-prefijo(L,[X|M]).

subLista(L,[X|_M]):-lista(X),subLista(L,X).

subLista(L,[_X|M]):-subLista(L,M).

% Determina si dos elementos son consecutivos

% Ejemplo: consecutivo([1,3,4,6,5,7,1,2],6,5). Yes

% consecutivo([1,3,4,6,5,7,1,2],5,6). No

consecutivo([_X,_Y|_Xs],_X,_Y).

consecutivo([_X|Xs],N,Z):-consecutivo(Xs,N,Z).
```

% BUSQUEDA DE ELEMENTOS

```
% Obtener el primer elemento de la lista
% Ejemplo: primerElem([3,2,4],X). X=3
primerElem([_X|_Y],_X).
% Obtener el ultimo elemento de la lista
% Ejemplo: ultimoElem([3,2,4],X). X=4
ultimoElem(L,S):-invertir(L,T),primerElem(T,S).
% Devuelve el enesimo elemento
% Ejemplo: enesimoElem([1,4,3,2,5],2,X). X=4
enesimoElem([],_N,[]):-!.
enesimoElem([_X|_Y],1,_X):-!.
enesimoElem([_X|Y],N,E):-N1 is N - 1,enesimoElem(Y,N1,E).
% Devuelve una lista con el elemento que se encuentra en la enesima posicion
% Ejemplo: nPosicion([1,4,2,3],2,X). X=[4]
% nPosicion([1,4,2,3],8,X). X=[]
nPosicion([],_N,[]):-!.
nPosicion([_X|_N],1,[_X]):-!.
nPosicion([_X|R],N,S):-M is N-1,nPosicion(R,M,S).
% Devuelve el numero de posicion de la primera ocurrencia de X
% Ejemplo: xPosicion(4,[7,2,1,8,3,6],X). X=0
% xPosicion(3,[7,2,1,3,4,3],X). X=4
xPosicion(X,L,S):-pertenece(X,L),!,xBusca(X,L,S).
xPosicion(X,L,S):-not(pertenece(X,L)),S is 0.
xBusca(_X,[],0):-!.
xBusca(_X,[_X|_M],1):-!.
xBusca(X,[_Y|M],S):-xBusca(X,M,T),S is T +1.
% Determina si un elemento X pertenece a la lista L (1 nivel)
```

% Ejemplo: pertenece(1,[3,4,6,1,3]). Yes pertenece(1,[3,[4,6,1],3]). No

pertenece(_X,[_X|_Y]).

pertenece(X,[_C|Y]):-pertenece(X,Y).

% Determina si un elemento X pertenece a la lista L (multinivel)

% Ejemplo: perteneceM(1,[3,4,6,1,3]). Yes

% perteneceM(1,[3,[4,6,1],3]). Yes

perteneceM(N,L):-listaAtomos(L,La),pertenece(N,La).

3

% ELIMINACION O REEMPLAZOS DE ELEMENTOS EN UNA LISTA

% Elimina los N primeros elementos de una lista y devuelve el resto

% Ejemplo: sacaNpri([1,3,5,1,2],2,L). L=[5,1,2]

sacaNpri([],_N,[]):-!.

sacaNpri([_X|_M],1,_M):-!.

sacaNpri([_X|M],N,S):-N1 is N - 1,sacaNpri(M,N1,S).

% Elimina los N ultimos elementos de una lista y devuelve el resto.

% Ejemplo: sacaNult([1,2,3,4,5],2,L). L = [1,2,3]

sacaNult(L,N,R):-invertir(L,L1),sacaNpri(L1,N,R1),invertir(R1,R).

% Elimina el elemento X de la lista en el 1º nivel

% Ejemplo: eliminaX([1,2,3,4,5],3,L). L=[1,2,4,5]

% eliminaX([1,2,[3,4],5,3],3,L). L=[1,2,[3,4],5]

eliminaX([],_X,[]):-!.

eliminaX([X|M],X,Z):- eliminaX(M,X,Z),!.

eliminaX([R|M],X,[R|Z]):- eliminaX(M,X,Z),!.

% Elimina el elemento X de la lista en todos los niveles

% Ejemplo: eliminaMx([1,2,[3,4],5,3],3,L). L=[1,2,[4],5]

% eliminaMx([2,[3,4],5,3],4,L). L=[2,[3],5,3]

```
eliminaMx([],_X,[]):-!.

eliminaMx([X],X,[]):-!.

eliminaMx([X|M],X,S):-eliminaMx(M,X,S),!.

eliminaMx([R|M],X,S):-lista(R),eliminaMx(R,X,T),eliminaMx(M,X,P),concatenar([T],P,S),!.

eliminaMx([R|M],X,S):-eliminaMx(M,X,T),concatenar([R],T,S).

% Elimina todos los elementos de la lista 2 que estan en la 1

% Ejemplo: elim12([1,4,3,2,7,9],[2,1],L). L=[]

% elim12([2,1],[1,4,3,2,7,9],L). L=[4,3,7,9]

% elim12([1,4],[1,2,[4,6],5,3],L). L=[2,[6],5,3]

elim12([],L,L):-!.

elim12([X|M],L,S):-eliminaMx(L,X,T),elim12(M,T,S).

% Elimina los elementos repetidos que estan en una lista, dejando el que primero ocurre

% Ejemplo: eliminaR([2,[3,4],5,3],L). L=[2,[3,4],5]

% eliminaR([2,[3,4],1,5,1,3],L). L=[2,[3,4],1,5]

eliminaR([],[]):-!.

eliminaR([X|M],S):-not(lista(X)),eliminaX(M,X,T),eliminaR(T,Y),concatenar([X],Y,S).

eliminaR([X|M],S):-lista(X),elim12(X,M,T),eliminaR(X,Y),

eliminaR(T,J),concatenar([Y],J,S).

% Elimina el primer elemento X que aparece en la lista.

% Ejemplo: eliminarPri(2,[4,2,1,2,1],X). X=[4,1,2,1]

% eliminarPri(3,[4,2,1,3,1],X). X=[4,2,1,1]

eliminarPri(_X,[],[]):-!.

eliminarPri(_X,[_X|_M],_M):-!.

eliminarPri(X,[_R|M],[_R|L]):-eliminarPri(X,M,L).

% Elimina el elemento que se encuentra en la enesima posicion

% Ejemplo: xElimina([1,2,3],2,L). L=[1,3]
```

% xElimina([1,2,3],5,L). No

xElimina([_X|_Y],1,_Y):-!.

xElimina([_X|Y],N,[_X|R]):-N1 is N - 1, xElimina(Y,N1,R).

% insertar X en la posicion N dentro de una Lista. N (L,N,X,R)

% Ejemplo: insertarXenN([1,2,3,4],1,5,R). R=[5,2,3,4]

insertarXenN([_C|L],1,X,[X|L]):-!.

insertarXenN([C|L],N,X,[C|R]):-N1 is N-1, insertarXenN(L,N1,X,R).

4

% Reemplaza la aparicion de un elemento X en una lista,en todos los niveles , por otro elemento Y.

% Ejemplo: xReemplazar(1,2,[1,2,1,4],X). X=[2,2,2,4]

% xReemplazar(1,2,[1,7,[1,3,6,9],1,4],X). X=[2,7,[2,3,6,9],2,4]

xReemplazar(_X,_Y,[],[]):-!.

xReemplazar(X,Y,[X|M],[Y|Z]):-xReemplazar(X,Y,M,Z),!.

xReemplazar(X,Y,[L|M],Z):-

xReemplazar(X,Y,L,T),xReemplazar(X,Y,M,R),!,concatenar([T],R,Z).

xReemplazar(X,Y,[L|M],[L|Z]):-xReemplazar(X,Y,M,Z),!.

% Devuelve la lista original sin el ultimo elemento

% Ejemplo: qu([1,2,3,4],L). L=[1,2,3]

qu(L,S):-long(L,Long),Aux is Long - 1, nPrimeros(L,Aux,S).

% Devuelve la lista original sin el primer elemento.

% Ejemplo: qp([1,2,3],L). L=[2,3]

qp(L,S):-sacaNpri(L,1,S).

% CREACION DE LISTAS

% concatenar dos listas

% Ejemplo: concatenar([1,2],[3,4],X). X=[1,2,3,4]

concatenar([],L,L):-!.

```
concatenar([X|M],L,[X|Y]):-concatenar(M,L,Y).

% concatena un Elemento a una Lista

% Ejemplo: concatenarElem([1,3,4],2,L). L=[1,3,4,2]

concatenarElem([],L,[L]).

concatenarElem([X|Xs],Y,[X|K]):-concatenarElem(Xs,Y,K).

% Devuelve todas las sublistas posibles en L de la lista pasada como primer argumento.

% Ejemplo: subSec([1,2,4],L). L=[1] [1,2] [1,2,4] [2] [2,4] [4]

subSec(M,L):-prefijo2(L,M).

subSec([_X|M],L):-subSec(M,L).

prefijo2([_X],[_X|_L]).

prefijo2([_X|Xs],[_X|L]):-prefijo2(Xs,L).

% Lista Atomos: genera una lista de atomos a partir de una lista anidada

% Ejemplo: listaAtomos([7,2,[casa,1],a],L). L=[7,2,casa,1,a]

atomo(X):-not(lista(X)).

listaAtomos([],[]).

listaAtomos([X|Y],[X|Ys]):- atomo(X),listaAtomos(Y,Ys).

listaAtomos([X|Y],Ys):- not(atomo(X)),listaAtomos(X,AtomosX),

concatenar(AtomosX,Y,Z),listaAtomos(Z,Ys).

% Da como resultado los N primeros elementos de una lista

% Ejemplo: nPrimeros([1,2,3,4,5],2,L). L=[1,2]

% nPrimeros([1,2,[4,6],5,3],3,L). L=[1,2,[4,6]]

nPrimeros(_L,0,[]):-!.

nPrimeros([],_N,[]):-!.

nPrimeros([_X|_M],1,[_X]):-!.

nPrimeros([X|M],N,S):-N1 is N - 1,nPrimeros(M,N1,T),concatenar([X],T,S).

% Devuelve todos los resultados posibles
```

```
% Ejemplo: nPrimerosT([1,2,3],L). L=[1] [1,2] [1,2,3]

nPrimerosT([_X|_M],[_X]).

nPrimerosT([X|M],L):-nPrimerosT(M,T),concatenar([X],T,L).
```

5

```
% Da como resultado los N ultimos elementos de una lista

% Ejemplo: nUltimos([1,2,3,4,5],2,L). L=[4,5]

% nUltimos([1,2,[4,6],5,3],3,L). L=[[4,6],5,3]

nUltimos(L,N,S):-invertir(L,T),nPrimeros(T,N,R),invertir(R,S).

% Arma una lista con todos los elementos menores que X

% Ejemplo: xMenores(3,[2,3,1,7,0],L). L=[2,1,0]

xMenores(_X,[],[]):-!.

xMenores(X,[Y|W],[Y|Z]):-X>Y,xMenores(X,W,Z),!.

xMenores(X,[_Y|W],Z):-xMenores(X,W,Z),!.

% Arma una lista con todos los elementos mayores que X

% Ejemplo: xMayores(3,[2,3,1,7,0],L). L=[7]

xMayores(_X,[],[]):-!.

xMayores(X,[Y|W],[Y|Z]):-X<Y,xMayores(X,W,Z),!.

xMayores(X,[_Y|W],Z):-xMayores(X,W,Z),!.

% Devuelve una lista con los anteriores elementos a X (primera ocurrencia)

% Ejemplo: ant(4,[3,2,1,4,1,6],L). X=[3,2,1]

% ant(6,[5,6,9,4,8,7],L). X=[5]

ant(_X,[],[]):-!.

ant(_X,[_X|_Y],[]):-!.

ant(X,[C|Y],[C|L]):-pertenece(X,[C|Y]),!,ant(X,Y,L).

% Devuelve una lista con los siguientes elementos a X (primera ocurrencia)

% Ejemplo: sig(4,[1,4,2,4,7],L). L=[2,4,7]
```

```
% sig(8,[1,4,2,1,7],L). No

sig(_X,[],[]):-!.

sig(_X,[_X|_Y],_Y):-!.

sig(X,[C|Y],L):-pertenece(X,[C|Y]),!,sig(X,Y,L).

% Devuelve dos listas, una con los elementos anteriores y otra con los siguientes a un valor X

% Ejemplo: izq_der_x(3,[1,3,2,6,4],I,D). I=[1] D=[2,6,4]

% izq_der_x(4,[1,3,2,6,4],I,D). I=[1,3,2,6] D=[]

% izq_der_x(1,[1,3,2,6,4],I,D). I=[] D=[3,2,6,4]

izq_der_x(X,L,I,D):-ant(X,L,I),sig(X,L,D).

% Invierte la lista en el primer nivel

% Ejemplo: invertir([3,2,[1,6],7,4],X). X = [4,7,[1,6],2,3]

invertir([],[]):-!.

invertir([X],[X]):-!.

invertir([X|M],L):-invertir(M,S),concatenar(S,[X],L).

% Invierte una lista en todos sus niveles

% Ejemplo: invertirM([3,2,[1,6],7,4],X). X = [4,7,[6,1],2,3]

invertirM([],[]):-!.

invertirM([X|M],S):-lista(X),invertirM(X,L),invertirM(M,T),concatenar(T,[L],S).

invertirM([X|M],S):-invertirM(M,T),concatenar(T,[X],S),!.

% Obtener una lista de los elementos mayores que X y otra con los menores que X

% Ejemplo: xMayMen(3,[2,3,1,7,0],L1,L2). L1=[2,1,0] L2=[7]

xMayMen(X,L,Men,May):-xMenores(X,L,Men),xMayores(X,L,May).

% Arma una lista con todas las posiciones de X en L.

% Ejemplo: ocurrencias(3,[1,3,2,4,65,7,3],L). L=[2,7]

ocurrencias(X,Y,Z):-invertir(Y,M),secOcurrencias(X,M,_R,L),invertir(L,Z),!.

secOcurrencias(_X,[_X],1,[1]):-!.
```

```
secOcurrencias(_X,[_Y],1,[]):-!.

secOcurrencias(X,[X|Xs],R,[R|Zs]):-secOcurrencias(X,Xs,R1,Zs),R is R1 + 1.

secOcurrencias(X,[_Y|Ys],R,Z):-secOcurrencias(X,Ys,R1,Z),R is R1 + 1 .
```



```
% Arma una lista con todos los elementos en secuencia creciente a partir del valor de X

% si en el recorrido encuentra un elemento decreciente (aunque sea mayor a X) no lo incluye

% Ejemplo: xCrece(4,[1,2,3,4,6,5],L). L=[4,6]

% xCrece(4,[1,2,3,4,5,6],L). L=[4,5,6]

xCrece(_X,[],[]).

xCrece(X,[Y|M],[Y|S]):-X=<Y,!,xCrece(Y,M,S).

xCrece(X,[_Y|M],S):-xCrece(X,M,S).

% Arma una lista con todos los elementos en secuencia decreciente hasta el valor de X

% si en el recorrido encuentra un elemento creciente (aunque sea menor a X) no lo incluye

% Ejemplo: xDecrece(4,[3,2,1,4,5,6],L). L=[3,2,1]

% xDecrece(4,[1,2,3,4,5,6],L). L=[1]

xDecrece(_X,[],[]).

xDecrece(X,[Y|M],[Y|S]):-X>=Y,!,xDecrece(Y,M,S).

xDecrece(X,[_Y|M],S):-xDecrece(X,M,S).

% Devuelve todos los elementos comunes a dos listas

% Ejemplo: interseccion([1,6,3],[2,1,8,3],L). L=[1,3]

interseccion(_L,[],[]):-!.

interseccion([],_L,[]):-!.

interseccion([_X|L],[_X|H],[_X|Z]):-interseccion(L,H,Z),!.

interseccion([X|L],[R|H],[X|Z]):-pertenece(X,H),eliminarPri(X,[R|H],L2),

interseccion(L,L2,Z),!.

interseccion([_X|L],[R|H],Z):-interseccion(L,[R|H],Z),!.
```

% MATEMATICAS CON LISTAS

```
% Suma los elementos de la lista (lista con elementos numericos)
% Ejemplo: sumaElem([3,2,2],X). X=7
sumaElem([X],X):-!.
sumaElem([X|Y],S):-sumaElem(Y,T),S is T + X.
% Suma los elementos respectivos de dos listas, generando otra con los resultados.
% Ambas listas deben tener el mismo tamano.
% Ejemplo: sumaListas([1,7,4],[9,3,6],L). L=[10,10,10]
sumaListas(L1,L2,L):-long(L1,S),long(L2,S),sumaLA(L1,L2,L).
sumaLA([],[],[]):-!.
sumaLA([X|Xs],[Y|Ys],[S|L]):-sumaLA(Xs,Ys,L),S is X + Y.
% Resta los elementos respectivos de dos listas, generando otra con los resultados.
% Ambas listas deben tener el mismo tamano.
% Ejemplo: restaListas([1,7,4],[9,3,6],L). L=[-8,4,-2]
restaListas(L1,L2,L):-long(L1,S),long(L2,S),restaLA(L1,L2,L).
restaLA([],[],[]):-!.
restaLA([X|Xs],[Y|Ys],[S|L]):-restaLA(Xs,Ys,L),S is X - Y.
% Multiplicacion de un numero por cada elemento de una lista.
% Ejemplo: multiplicaEscalar(2,[1,2,3,4],L). L=[2,4,6,8]
multiplicaEscalar(_,[],[]):-!.
multiplicaEscalar(N,[X|L],[Z|Zs]):-multiplicaEscalar(N,L,Zs),Z is X*N.
% Producto Cartesiano de dos listas.
% Ejemplo: productoCA([1,2,3],[2,3,4],P). P=20.
productoC(L1,L2,S):-long(L1,L),long(L2,L),productoCA(L1,L2,S).
productoCA([],[],0):-!.
```

```
productoCA([X|Xs],[Y|Ys],S):-productoCA(Xs,Ys,S2), S3 is X*Y, S is S2+S3.

% Determina cuantas veces se repite X en una lista.

% Ejemplo: ocurre(1,[3,2,1,6,1],X). X=2

% ocurre(1,[3,2,6],X). X=0

ocurre(_X,[],0).

ocurre(X,[X|R],N):- ocurre(X,R,N1), N is N1 +1,!.

ocurre(X,[_Y|R],N):- ocurre(X,R,N).

% Determina cuantas veces se repite X en una lista (Multinivel)

% Ejemplo: ocurreM(1,[3,2,[1,3,5],1,[6,2,1]],S). S=3

ocurreM(X,L,S):-listaAtomos(L,L1),ocurre(X,L1,S).

% Verifica si M aparece solo una vez en L (primer nivel).

% Ejemplo: unico([1,2],[3,2,[1,2],7]). Yes

% unico(7,[3,2,[1,2],6]). No

unico(M,L):-ocurre(M,L,N),N = 1.

% Verifica si M aparece más de una vez en L (primer nivel).

% Ejemplo: +de1(2,[3,2,[1,2],7]). No

% +de1(2,[3,2,[1,2],6,2]). Yes

+de1(M,L):- ocurre(M,L,N),N > 1.

% Calcula la cantidad de elementos iguales que se encuentan en la misma posicion en dos listas

% ejemplo: elemIguales([4,5,1],[1,2,3],X). X=0

% elemIguales([1,5,1],[1,2,3],X). X=1

% Si un elemento se repite en otra posicion, no se cuenta.

elemIguales(_L,[],0):-!.

elemIguales([],_L,0):-!.

elemIguales([_X],[_X],1):-!.
```

```
elemIguales([X|Y],[X|Z],S):-elemIguales(Y,Z,T),!,S is T + 1.

elemIguales([_X|Y],[_R|Z],S):-elemIguales(Y,Z,S).

% Cuenta los elementos que se encuentran en dos listas en diferente posiciones

% Ejemplo: examinaLista([4,5,1],[1,2,3],X,Y). X=1 Y=0

% examinaLista([1,5,1],[1,2,3],X,Y). X=1 Y=1

% En X devuelve cantidad de elementos iguales en diferente posicion.

% En Y devuelve cantidad de elementos iguales en la misma posicion.

elemIgualesDif([],_L,_N,0).

elemIgualesDif([X|Y],L,P,S):-xElimina(L,P,L1),ocurre(X,L1,S1),P1 is P + 1,

elemIgualesDif(Y,L,P1,R),S is S1 + R.

examinaLista(L1,L2,D,S):-elemIgualesDif(L1,L2,1,D),elemIguales(L1,L2,S).
```

% METODOS DE ORDENAMIENTO

```
% Dada una lista nos dice si esta o no ordenada ascendentemente

% Ejemplo: ordenada([1,2,3,4,5]). Yes

% ordenada([1,2,7,4,5]). No

ordenada([_X]).

ordenada([X,Y|Ys]):-X=<Y,ordenada([Y|Ys]).

% Genera todas las permutaciones de una lista

% Ejemplo: permutacion([2,3,1],L). L=[2,3,1] L=[2,1,3]

% L=[3,2,1] L=[3,1,2]

% L=[1,2,3] L=[1,3,2]

permutacion([],[]):-!.

permutacion(Xs,[Z|Zs]):-desarma(Z,Xs,Ys),permutacion(Ys,Zs).

desarma(X,[X|Xs],Xs).

desarma(X,[Y|Ys],[Y|Zs]):-desarma(X,Ys,Zs).

% Ordena la lista ascendentemente por Permutacion
```

```
% Ejemplo: ordenaP([1,3,2],L). L=[1,2,3]

% ordenaP([1,3,2,6,2],L). L=[1,2,2,3,6]

ordenaP(X,Y):-permutacion(X,Y),ordenada(Y),!.

% Ordenación Quicksort

% Ejemplo: quickSort([1,4,3,2,5,7,6],L). L=[1,2,3,4,5,6,7]

quickSort([],[]).

quickSort([X|Xs],Y):-

xMayMen(X,Xs,Li,Ld),quickSort(Li,LI),quickSort(Ld,LD),concatenar(LI,[X|LD],Y).

% Insercion ordenada de un elemento en una lista ordenada

% Ejemplo: insertar(3,[1,4,5,7],L). L=[1,3,4,5,7]

insertar(X,[],[X]).

insertar(X,[Y|Ys],[Y|Zs]):-X>Y,insertar(X,Ys,Zs).

insertar(X,[Y|Ys],[X,Y|Ys]):-X=<Y.

% Ordenacion de lista en forma ascendente

% Ejemplo: ordenar2([1,2,5,3,3,6,8,7],L). L=[1,2,3,3,5,6,7,8]

ordenar2([],[]).

ordenar2([X|Xs],Ys):-ordenar2(Xs,Zs),insertar(X,Zs,Ys).

% Ordena una lista en orden descendente

% Ejemplo: ordenDescen([1,4,3,2],L). L=[4,3,2,1]

ordenDescen(L,L2):-ordenar2(L,L3),invertir(L3,L2).
```



```
% MANIPULACION DE ARBOLES BINARIOS

% Representacion de arboles binarios:

% arbol(nodo,nil,nil) <= unicamente la raiz.

% arbol(nodo,arbol(hoja,nil,nil),arbol(hoja,nil,nil) <= raiz y 2 hojas. a raiz / nodo

% / \
```

```
% Determina si un arbol esta vacio b c hojas
% Ejemplo: vacioA(arbol(a,nil,arbol(b,nil,nil))). No
% vacioA(arbol(nil)). Yes
vacioA(arbol(nil)):-!.
% Arma una lista con todos los elementos del arbol
% Ejemplo: listaArbol(arbol(a,arbol(b,nil,nil),arbol(c,nil,nil)),L). L=[a,b,c]
listaArbol(arbol(A,nil,nil),[A]):-!.
listaArbol(arbol(A,X,nil),S):-listaArbol(X,P),concatenar([A],P,S).
listaArbol(arbol(A,nil,X),S):-listaArbol(X,P),concatenar([A],P,S).
listaArbol(arbol(A,X,Y),S):-listaArbol(X,P),listaArbol(Y,R),
concatenar([A],P,U),concatenar(U,R,S).
% Determina si un elemento pertenece al arbol
% Ejemplo: perteneceA(arbol(a,arbol(b,nil,nil),arbol(c,nil,nil)),a). Yes
% perteneceA(arbol(a,arbol(b,nil,nil),arbol(c,nil,nil)),z). No
perteneceA(arbol(_A,_X,_Y),_A):-!.
perteneceA(arbol(_A,X,_Y),B):-perteneceA(X,B),!.
perteneceA(arbol(_A,_X,Y),B):-perteneceA(Y,B),!.
% cuenta la cantidad de elementos que tiene un arbol
% Ejemplo: cantElemA(arbol(a,arbol(b,nil,nil),arbol(c,nil,nil)),C). C=3
cantElemA(arbol(_A,nil,nil),1):-!.
cantElemA(arbol(_A,X,nil),N):-cantElemA(X,K),N is K + 1 .
cantElemA(arbol(_A,nil,X),N):-cantElemA(X,K),N is K + 1 .
cantElemA(arbol(_A,X,Y),N):-cantElemA(X,K),cantElemA(Y,T), N is 1 + K + T.
% Determina si una arbol binario es completo
% Ejemplo: completoA(arbol(a,arbol(b,nil,nil),arbol(c,nil,nil))). Yes
% completoA(arbol(a,arbol(b,nil,nil),nil)). No
```

```
completoA(arbol(_A,nil,nil)):-!.

completoA(arbol(_A,X,Y)):-profundidadA(X,N),profundidadA(Y,M),N =:= M,

completoA(X),completoA(Y).

% Calcula la profundidad del arbol (la raiz tiene nivel 0)

% Ejemplo:
profundidadA(arbol(a,arbol(b,nil,nil),arbol(c,arbol(e,arbol(f,nil,nil),arbol(j,nil,arbol(z,nil,nil))),nil)),P
).

% P = 4

profundidadA(arbol(_A,nil,nil),0):-!.

profundidadA(arbol(_A,Y,nil),N):-profundidadA(Y,B),N is B + 1,!.

profundidadA(arbol(_A,nil,X),N):-profundidadA(X,B),N is B + 1,! .

profundidadA(arbol(_A,X,Y),N):-profundidadA(X,B),profundidadA(Y,C), B1 is B+1,

C1 is C+1,mayor(B1,C1,N),!.

mayor(B,C,C):-C>=B,!.

mayor(_B,_C,_B).

% Recorre un arbol en preorden

% Ejemplo: preordenA(arbol(a,arbol(b,nil,nil),arbol(c,nil,nil)),L). L=[a,b,c]

preordenA(arbol(A,nil,nil),[A]):-!.

preordenA(arbol(A,X,nil),[A|S]):-preordenA(X,S),!.

preordenA(arbol(A,nil,X),[A|S]):-preordenA(X,S),!.

preordenA(arbol(A,X,Y),[A|S]):-preordenA(X,T),preordenA(Y,O),concatenar(T,O,S).
```



```
% Recorre un arbol en inorden

% Ejemplo: inordenA(arbol(a,arbol(b,nil,nil),arbol(c,nil,nil)),L). L=[b,a,c]

inordenA(arbol(A,nil,nil),[A]):-!.

inordenA(arbol(A,X,nil),S):-inordenA(X,C),concatenar(C,[A],S).
```

```
inordenA(arbol(A,nil,X),[A|S]):-inordenA(X,S).

inordenA(arbol(A,X,Y),S):-inordenA(X,C),inordenA(Y,F),

concatenar(C,[A],D),concatenar(D,F,S).

% Recorre un arbol en postorden

% Ejemplo: postordenA(arbol(a,arbol(b,nil,nil),arbol(c,nil,nil)),L). L=[b,c,a]

postordenA(arbol(A,nil,nil),[A]):-!.

postordenA(arbol(A,X,nil),S):-postordenA(X,C),concatenar(C,[A],S).

postordenA(arbol(A,nil,X),S):-postordenA(X,C),concatenar(C,[A],S).

postordenA(arbol(A,X,Y),S) :-postordenA(X,C),postordenA(Y,F),

concatenar(C,F,D),concatenar(D,[A],S).

% Devuelve el numero de hijos en un arbol binario. Para arboles con solo la raiz, se cuenta 1 hijo
(la misma raiz)

% Ejemplo: cuentaHoja(arbol(a,arbol(b,nil,nil),arbol(e,nil,arbol(k,nil,nil))),X). X=2

% cuentaHoja(arbol(a,nil,nil),X). X=1

cuentaHoja(arbol(_R,nil,nil),1):-!.

cuentaHoja(arbol(_R,Hi,nil),S):-cuentaHoja(Hi,S).

cuentaHoja(arbol(_R,nil,Hd),S):-cuentaHoja(Hd,S).

cuentaHoja(arbol(_R,Hi,Hd),S):-cuentaHoja(Hi,S1),cuentaHoja(Hd,S2),S is S1+S2.

% version 2

cuentaHoja2(nil,0):-!.

cuentaHoja2(arbol(_R,nil,nil),1):-!.

cuentaHoja2(arbol(_R,Hi,Hd),S):-cuentaHoja2(Hi,S1),cuentaHoja2(Hd,S2),S is S1 + S2.

% Suma el valor de todas las hojas (valores numericos)

% Ejemplo: sumaHoja(arbol(1,nil,arbol(2,arbol(4,nil,nil),arbol(3,nil,nil))),X). X=7

sumaHoja(nil,0):-!.

sumaHoja(arbol(R,nil,nil),R):-!.
```

sumaHoja(arbol(_R,Hi,Hd),S):-sumaHoja(Hi,S1),sumaHoja(Hd,S2),S is S1 + S2.

% Extrae un arbol a partir de un nodo X dado.

% Ejemplo: subArbol(b,arbol(a,arbol(b,arbol(c,nil,nil),arbol(d,nil,nil)),arbol(e,nil,nil)),A).

% A = arbol(b,arbol(c,nil,nil),arbol(d,nil,nil))

% subArbol(b,arbol(a,arbol(b,arbol(c,nil,nil),nil),nil),A).

% A = arbol(b,arbol(c,nil,nil),nil)

% subArbol(j,arbol(a,arbol(b,nil,nil),arbol(c,arbol(e,arbol(f,nil,nil),arbol(j,nil,arbol(z,nil,nil))),nil)),A).

% A = arbol(j,nil,arbol(z,nil,nil))

subArbol(X,arbol(X,HI,HD),arbol(X,HI,HD)):-!.

subArbol(X,arbol(_Y,HI,_HD),R):-subArbol(X,HI,R).

subArbol(X,arbol(_Y,_HI,HD),R):-subArbol(X,HD,R).

% MANIPULACION DE GRAFOS

% Representacion *[[a,b],[b,a],[b,c]]* es:

%

% a <--> b -->c

%

% En los grafos no dirigidos se deben especificar los dos pares de nodos.

% Determina si existe un camino entre X e Y. Devuelve True o False.

% Ejemplo: camino(c,b,[[a,b],[b,c],[d,e],[c,d],[b,e],[e,c],[e,f],[a,a]]). No

% camino(c,f,[[a,b],[b,c],[d,e],[c,d],[b,e],[e,c],[e,f],[a,a]]). Yes

camino(X,Y,G):-caminoAux(X,Y,G,[]).

caminoAux(X,Y,G,_T):-pertenece([X,Y],G).

caminoAux(X,Y,G,T) :-pertenece([X,Z],G),not(pertenece([X,Z],T)),

concatenar([[X,Z]],T,Tt),caminoAux(Z,Y,G,Tt).

% Devuelve la lista R con los nodos del grafo G.

```
% Ejemplo: nodos([[a,b],[b,c],[d,e],[c,d],[b,e],[e,c],[e,f],[a,a]],R). R=[a, b, c, d, e, f]
nodos(G,R):-listaAtomos(G,L),eliminaR(L,R).
% Cuenta los nodos diferentes del grafo
% Ejemplo: cuentaNodos([[a,b],[b,c],[d,e],[c,d],[b,e],[e,c],[e,f],[a,a]],R). R=6
cuentaNodos(G,R):-nodos(G,L),long(L,R).
% Para grafos no dirigidos. Determina si un Grafo es o no conexo.
% Ejemplo: conexo([[a,b],[b,a],[b,c],[c,b],[c,d],[d,c]]). Yes
% conexo([[a,b],[b,a],[b,c],[c,b],[c,d]]). No
conexo(G):-nodos(G,R),conexoAux(R,G).
conexoAux([],_G):-!.
conexoAux([X|Xs],G):-nodos(G,N),eliminaX(N,X,R),llegaTodos(X,R,G),conexoAux(Xs,G).
llegaTodos(_X,[],_G):-!.
llegaTodos(X,[Y|Ys],G):-camino(X,Y,G),llegaTodos(X,Ys,G).
% Como parámetro se le pasa un camino, un nodo y un grafo.
% Ejemplo: visita_nodo([a,b,c],b,[[a,b],[b,c],[c,d]]). Yes
% visita_nodo([a,b,c],d,[[a,b],[b,c],[c,d]]). No
visita_nodo(C,N,G):-pertenece(N,C),esCamino(C,G).
% El primer parámetro es una lista indicando un camino. El segundo parámetro
% es el grafo. Responde Yes si existe en el grafo el camino especificado.
% Ejemplo: esCamino([a,b,c],[[a,b],[b,c],[c,d]]). Yes
% esCamino([a,c],[[a,b],[b,c],[c,d]]). No
esCamino([],_G).
esCamino([X],G):-nodos(G,L),pertenece(X,L).
esCamino([X,Y|Z],G):-pertenece([X,Y],G),esCamino([Y|Z],G).
% Devuelve la longitud de un camino. Se le pasa el inicio y el final.
% Ejemplo: caminoLongL(a,c,R,[[a,b],[b,c],[c,d]]). R=2
```

```
caminoLongL(X,Y,L,Go):-caminoLongAux(X,Y,L,Go,[]).

caminoLongAux(X,Y,1,Go,_T):-pertenece([X,Y],Go).

caminoLongAux(X,Y,L,Go,T) :-pertenece([X,Z],Go),not(pertenece([X,Z],T)),

concatenar([[X,Z]],T,Ti),caminoLongAux(Z,Y,H,Go,Ti),

L is H + 1.

% Dada una longitud de camino y un grafo, devuelve otro grafo compuesto por pares de nodos

% cuya distancia entre ellos sea la longitud dada en el grafo original (Ejercicio 8, guia 3).

% Ejemplo: generarGrafo(1,[[a,b],[b,c],[c,d],[b,e]],G). G=[[a,b],[b,c],[b,e],[c,d]]

% generarGrafo(2,[[a,b],[b,c],[c,d],[b,e]],G). G=[[a,c],[a,e],[b,d]]

% generarGrafo(3,[[a,b],[b,c],[c,d],[b,e]],G). G=[[a,d]]

% generarGrafo(4,[[a,b],[b,c],[c,d],[b,e]],G). G=[]

% Nota: si la longitud es 1, devuelve el grafo original.

generarGrafo(L,Go,Gr):-nodos(Go,R),ggAux(R,R,L,Go,Gr),!.

ggAux([X],R,L,Go,M) :-lCaminos(X,R,L,Go,M).

ggAux([X|Y],R,L,Go,B):-lCaminos(X,R,L,Go,A),ggAux(Y,R,L,Go,B1),

concatenar(A,B1,B).

lCaminos(X,[Y],L,Go,[[X,Y]]):-caminoLongL(X,Y,L,Go).

lCaminos(_X,[_Y],_L,_Go,[]).

lCaminos(X,[A|B],L,Go,[[X,A]|N]):-caminoLongL(X,A,L,Go),lCaminos(X,B,L,Go,N),!.

lCaminos(X,[_A|B],L,Go,M):-lCaminos(X,B,L,Go,M).

% Suma los nodos de un grafo. Estos deben ser numeros.

% Ejemplo: sumaNodos([[1,2],[3,4],[5,6],[2,3],[3,5]],S). S=21

sumaNodos(G,R):-nodos(G,L),sumaElem(L,R).
```



```
% Determina si existe un rizo en el grafo.

% Ejemplo: tieneRizo([[a,b],[b,c],[c,d],[a,a]]). Yes
```

```
% tieneRizo([[a,b],[b,c],[c,d]]). No

tieneRizo(G):-nodos(G,L),rizoAux(L,G).

rizoAux([X|_Xs],G):-pertenece([X,X],G).

rizoAux([_X|Xs],G):-rizoAux(Xs,G).

% 82 - Determina si existe al menos un bucle en el grafo dado.

% Ejemplo: tieneBucle([[a,b],[b,c],[c,d],[c,b]]). Yes

% tieneBucle([[a,b],[b,c],[c,d]]). No

tieneBucle([[X,Y]|G]):-camino(Y,X,G).

tieneBucle([[_X,_Y]|G]):-tieneBucle(G).

% Devuelve el camino más corto entre dos nodos.

% Ejemplo: mejorCamino(a,e,[[a,b],[b,c],[c,d],[b,d],[d,e],[c,e]],C). C=[a,b,d,e]

% mejorCamino(a,c,[[a,b],[b,c],[c,d],[b,d],[d,e],[c,e]],C). C=[a,b,c]

mejorCamino(I,F,G,C):-camino(I,F,G),mCaminoAux(I,F,G,C),esCamino(C,G),!.

mCaminoAux(I,F,G,[I,F]):-pertenece([I,F],G).

mCaminoAux(I,F,G,[I,C|Cs]):-mCaminoAux(C,F,G,[C|Cs]).
```

% MANIPULACION DE MATRICES

```
% Formato de matrices: lista de lista.

% 1 2 3

% Ejemplo para N=3 => [ [1,2,3] , [4,5,6] , [7,8,9] ] => 4 5 6

% 7 8 9

% Verifica que el argumento sea una matriz cuadrada.

% Ejemplo: matrizCuad([[1,2,3],[4,5,6],[7,8,9]]). Yes

% matrizCuad([[1,2,3],[4,5,6]]). No

matrizCuad(M):-long(M,S),matrizAux(M,S).

matrizAux([],_):-!.

matrizAux([M|Ms],S):-long(M,S),matrizAux(Ms,S).
```

```
% Verifica que el argumento sea una matriz.
% Ejemplo: esMatriz([[1,2,3],[2,5,4]]). Yes
% esMatriz([[1,2,3],[2,5]]). No
esMatriz([F|M]):-long(F,S),esMatrizA(M,S).
esMatrizA([],_):-!.
esMatrizA([F|M],S):-long(F,S),esMatrizA(M,S).
% Rota una matriz (rotacion antihoraria).
% Ejemplo: rotarA([[1,2,3],[4,5,6],[7,8,9]],M). M = [[3,6,9],[2,5,8],[1,4,7]]
%
% 1 2 3 3 6 9
% 4 5 6 => 2 5 8
% 7 8 9 1 4 7
rotarA([],[]):-!.
rotarA(M,[S|M2]):-armar1renglon(M,S,Ms),rotarA(Ms,M2),!.
rotarA(_,[]).
armar1renglon([],[],[]):-!.
armar1renglon([C|M],[U|R],[S|N]):-ultimoElem(C,U),qu(C,S),armar1renglon(M,R,N).
```



```
% Rota una matriz (rotacion horaria).
% Ejemplo: rotarH([[1,2,3],[4,5,6],[7,8,9]],M). M = [[7,4,1],[8,5,2],[9,6,3]]
%
% 1 2 3 7 4 1
% 4 5 6 => 8 5 2
% 7 8 9 9 6 3
rotarH([],[]):-!.
rotarH(M,[S|M2]):-armarUNrenglon(M,T,Ms),invertir(T,S),rotarH(Ms,M2),!.
```

```
rotarH(_,[]).

armarUNrenglon([],[],[]):-!.

armarUNrenglon([C|M],[U|R],[S|N]):-primerElem(C,U),qp(C,S),armarUNrenglon(M,R,N).

% Recorrido en espiral de una matriz (sentido horario).

% Ejemplo: espiralH([[1,2,3],[4,5,6],[7,8,9]],M). M = [1,2,3,6,9,8,7,4,5]

espiralH([],[]):-!.

espiralH([E1|M],L):-rotarA(M,R),espiralH(R,L1),concatenar(E1,L1,L).

% Recorrido en espiral de una matriz (sentido antihorario).

% Ejemplo: espiralA([[1,2,3],[4,5,6],[7,8,9]],M). M = [1,4,7,8,9,6,3,2,5]

espiralA([],[]):-!.

espiralA([E1|M],L):-

rotarH([E1|M],[E2|R]),espiralA(R,L1),invertir(E2,T),concatenar(T,L1,L).

% Igualdad de matrices

% Ejemplo: igualMatriz([[1,2,3],[4,5,6],[7,8,9]],[[1,2,3],[4,5,6],[7,8,9]]). Yes

% igualMatriz([[1,2,3],[4,5,6],[7,8,9]],[[1,2,3],[4,5,6],[7,9,8]]). No

igualMatriz(M1,M1).

% Suma matrices.

% Ejemplo: sumaMatrices([[1,2,3],[2,1,0],[0,0,1]],[[-1,2,1],[0,0,2],[1,2,7]],S). S = [[0,4,4],[2,1,2],
[1,2,8]]

% sumaMatrices([[1,2],[2,1],[0,0]],[[-1,2],[0,0],[1,2]],S). S = [[0,4],[2,1],[1,2]]

sumaMatrices([],[],[]):-!.

sumaMatrices([M1|Ms1],[M2|Ms2],[M|Ms]):-
sumaMatrices(Ms1,Ms2,Ms),sumaListas(M1,M2,M).

% Resta matrices.

% Ejemplo: restaMatrices([[1,2,3],[2,1,0],[0,0,1]],[[-1,2,1],[0,0,2],[1,2,7]],R). R = [[2,0,2],[2,1,-2],[-
1,-2,-6]]

% restaMatrices([[1,2],[2,1],[0,0]],[[-1,2],[0,0],[1,2]],R). R = [[2,0],[2,1],[-1,-2]]
```

```
restaMatrices([],[],[]):-!.

restaMatrices([M1|Ms1],[M2|Ms2],[M|Ms]):-restaMatrices(Ms1,Ms2,Ms),restaListas(M1,M2,M).

% Producto de matrices.

% Ejemplo: productoMatricial([[1,2,3],[2,1,0],[0,0,1]],[[-1,2,1],[0,0,2],[1,2,7]],M). M = [[2,8,26],[-2,4,4],[1,2,7]]

% productoMatricial([[1,2],[2,0],[0,0]],[[-1,2,1],[0,0,2]],M). M = [[-1,2,5],[-2,4,2],[0,0,0]]

% productoMatricial([[3,2,1,-2],[-6,4,0,3]],[[1],[4],[0],[2]],M). M = [[7],[16]]

% productoMatricial([[-4,5,1],[0,4,2]],[[3,-1,1],[5,6,4],[0,1,2]],M). M = [[13,35,18],[20,26,20]].

productoMatricial([F|Ms1],M2,M):-long(F,L),long(M2,L),

rotarA(M2,M22),productoMA([F|Ms1],M22,M).

productoMA([],_,[]):-!.

productoMA([F|Ms1],M2,[L1|Ls]):-productoMA(Ms1,M2,Ls),pMA(F,M2,L),invertir(L,L1).

pMA(_,[],[]):-!.

pMA(F,[M2|Ms2],[S|L]):-pMA(F,Ms2,L),productoC(F,M2,S).

% Extrae la diagonal principal de una matriz cuadrada.

% Ejemplo: diagonal([[1,2,3],[4,5,6],[7,8,9]],D). D=[1,5,9]

diagonal(M,D):-matrizCuad(M),diagonalA(M,1,D1),invertir(D1,D).

diagonalA([],_,[]):-!.

diagonalA([F|M],N,D):-N1 is N + 1,diagonalA(M,N1,D1),nPosicion(F,N,E),concatenar(D1,E,D).
```



```
% Matriz escalonada inferior en matrices cuadradas.

% Ejemplo: escalonadaI([[1,0,0],[4,5,0],[7,8,9]]). Yes

% escalonadaI([[1,0,1],[4,5,0],[7,8,9]]). No

%

% 1 0 0 1 0 1 Un matriz se dice escalonada cuando al pasar de un renglón a otro,

% 4 5 0 => Yes 4 5 0 => No la cantidad de 0 (ceros) se va incrementando.
```

```
% 7 8 9 7 8 9
escalonadaI(M):-matrizCuad(M),escalonadaIA(M,1,R),sumaElem(R,S),S=0.
escalonadaIA([],_,[]):-!.
escalonadaIA([F|M],P,L):-P1 is P+1,escalonadaIA(M,P1,L1),
sacaNpri(F,P,R),concatenar(R,L1,L).
% Matriz escalonada superior en matrices cuadradas.
% Ejemplo: escalonadaI([[1,0,0],[4,5,0],[7,8,9]]). Yes
% escalonadaI([[1,0,4],[4,5,0],[7,8,9]]). No
%
% 1 0 0 1 0 4
% 4 5 0 => Yes 4 5 0 => No
% 7 8 9 7 8 9
escalonadaS(M):-rotarH(M,M1),rotarH(M1,M2),escalonadaI(M2).
% Verifica si una matriz es la Matriz Identidad (matrices cuadradas).
% Ejemplo: identidad?([[1,0,0],[0,1,0],[0,0,1]]). Yes
% identidad?([[1,0,4],[4,5,0],[7,8,9]]). No
% identidad?([[1,0,0],[0,1,0],[0,1,1]]). No
identidad?([F|M]):-
escalonadaI([F|M]),escalonadaS([F|M]),long(F,R),diagonal([F|M],L),sumaElem(L,R).
% Multiplicar una matriz por un numero escalar.
% Ejemplo: multMatrizEscalar(2,[[1,1,1],[1,1,1],[1,1,1]],M). M = [[2,2,2],[2,2,2],[2,2,2]]
% multMatrizEscalar(2,[[4,5,6],[1,2,3]],M). M = [[8,10,12],[2,4,6]]
multMatrizEscalar(_,[],[]):-!.
multMatrizEscalar(X,[F|M],[L|Ls]):-multMatrizEscalar(X,M,Ls),multiplicaEscalar(X,F,L).
% Devuelve la diagonal n-esima de una matriz.
% Ejemplo: diagonalN([[1,2,3],[4,5,6],[7,8,9]],1,R). R = [1,5,9]
```

```
% diagonalN([[1,2,3],[4,5,6],[7,8,9]],2,R). R = [2,6]

diagonalN([],_,[]):-!.

diagonalN([M|Ms],N,[]):-long(M,L),L < N,!.

diagonalN([M|Ms],N,D):-N1 is N+1,diagonalN(Ms,N1,D1),

nPosicion(M,N,E),concatenar(E,D1,D).

% Devuelve la matriz original transpuesta ([i,j]=>[j,i]).

% Ejemplo: transpuesta([[1,2,3],[4,5,6],[7,8,9]],M). M = [[1,4,7], [2,5,8], [3,6,9]]

transpuesta(M,Mt):-rotarH(M,Mr),transpuestaAux(Mr,Mt).

transpuestaAux([X],[Y]):-invertir(X,Y),!.

transpuestaAux([X|Xs],[Y|Ys]):-invertir(X,Y),transpuestaAux(Xs,Ys).

% Dimensiones en matrices: matrizFil retorna las filas de una matriz,

% y matrizCol las columnas.

matrizFil(M,F):-long(M,F).

matrizCol([M|Ms],C):-long(M,C).

% Dada una matriz, busca el elemento en las coordenadas [x,y] dadas.

% Ejemplo: matrizXY([[1,2,3],[4,5,6],[7,8,9]],2,3,V). V=8

matrizXY(M,C,F,V):-matrizFil(M,F1),matrizCol(M,C1),F=<F1,C=<C1,

enesimoElem(M,F,A),enesimoElem(A,C,V).
```

% EJERCICIOS DE EXAMEN

```
% 2003

% Devuelve todas las sublistas de S cuya suma de elementos es igual a E

% Ejemplo: subSecR([1,2,3,4,5,6],3,L). L=[1,2] L=[3]

% subSecR([1,2,3,4,5,6],7,L). L=[3,4]
```

```
subSecR(S,E,R):-subSec(S,R),sumaElem(R,E).

% Ejercicio de examen. 28/02/2005

% Verifica si la suma de los elementos anteriores a aquel cuya posicion

% se especifica es igual al elemento cuya posicion se especifica.

% Ejemplo: sumaAnterior([1,7,8,16,5],4). Yes

% sumaAnterior([1,7,7,16,5],4). No (1 + 7 + 7 = 15 <> 16)

sumaAnterior(L,N):-nPrimerosT(L,L1),qu(L1,L2),sumaElem(L2,S),

enesimoElem(L,N,S).

% Diciembre-2004

% asignar(Lnum,Grafo,Restricciones,Lasignaciones).

% Lasignaciones es un predicado no-ground

% asignar([1,2,3,4,5,6,7,8,9],

% [[a,b],[b,c],[c,a],[a,e],[e,f],[f,c],[c,d],[b,d],[d,g],[g,d],[g,h],[h,f],[h,i],[f,i],[e,i]],

% [[a,b,c,12],[b,c,d,11],[a,c,e,f,28],[c,d,f,g,20],[i,f,e,22],[i,f,h,16],[f,g,h,13]],La).

%

% Dado un grafo, el valor de las restricciones surge de sumar los valores

% de los nodos que se vinculan. "La" devuelve una lista de nodos con los

% valores que deben tener para satisfacer los criterios.

%

% La=[[a,7],[b,1],[c,4],[e,9],[f,8],[d,6],[g,2],[h|...], [...|...]]

%

% version 1

asignar1(N,G,R,A):-nodos(G,No),permutacion(N,Np),

combinar1(No,Np,A),contemplar1(R,A),!.

contemplar1([],_).

contemplar1([R|Rs],A):-contemplarAux1(R,A),contemplar1(Rs,A).
```

```
contemplarAux1(R,A):-pR1(R,A,L),sumaElem(L,M),invertir(R,[H|_Hs]),H=M.

pR1(_R,[],[]).

pR1(R,[[X,_L]|As],[_L|Ls]):-pertenece(X,R),pR1(R,As,Ls).

pR1(R,[_A|As],L):-pR1(R,As,L).

combinar1([],[],[]).

combinar1([_N|Ns],[_M|Ms],[[_N,_M]|S]):-combinar1(Ns,Ms,S).

% version 2

asignar2(N,G,R,A):-nodos(G,No),permutacion(N,Np),combinar1(No,Np,A),validar2(R,A),!.

validar2([],_A).

validar2([R|Rs],A):-nUltimos(R,1,U),long(R,L),L1 is L-1,nPrimeros(R,L1,N),

validarAux2(N,U,A),validar2(Rs,A).

validarAux2(N,[S],A):-suma2(N,A,S).

suma2([E],A,S):-pertenece([E,S],A).

suma2([E|Es],A,S):-pertenece([E,S1],A),suma2(Es,A,S2),S is S1 +S2.
```

2

% TORTUGA (ninja ;) N=3

```
% 24/02/03

% Debe proporcionarse el tamano del tablero, las coordenadas (X,Y) de las celdas

% y el numero de pasos en que deben ser alcanzadas por la tortuga. En C se

% informa los caminos que se encuentren que satisfagan la restriccion (celda en

% en tantos pasos. Se asume que la tortuga no necesita recorrer todas las celdas.

%

% Ejemplo: tortuga1(4,[[2,1,1],[3,3,6]],C).

%

% 1 de tantos caminos: C=[[2,1,1],[1,1,2],[1,2,3],[2,2,4],[3,2,5],[3,3,6]]

% version 1
```

```
tortuga1(N,M,C):-celdaC(X,Y),moverC([X,Y,1],[X1,Y1,2],N),
tAux1(N,M,[[X,Y,1],[X1,Y1,2]],C).
tAux1(_N,M,C,C):-pertenece([1,1,_],C),verificaC(C,M),!.
tAux1(N,M,C1,C):-ultimoElem(C1,Cu),moverC(Cu,[X,Y,S],N),
not(pertenece([X,Y,_],C1)),concatenarElem(C1,[X,Y,S],Cn),
tAux1(N,M,Cn,C).
% version 2
tortuga2(N,M,C):-caminos2(N,C),verificaC(C,M).
caminos2(N,[[X,Y,1]|Z]):-celdaC(X,Y),caminoAux2(N,[[X,Y,1]|Z],[]).
caminoAux2(_N,[[1,1,_]],_):-!.
caminoAux2(N,[[X,Y,Z],[X1,Y1,Z1]|M],A):-moverC([X,Y,Z],[X1,Y1,Z1],N),
not(pertenece([X1,Y1,_],A)),
concatenar([[X,Y,Z]],A,A1),caminoAux2(N,[[X1,Y1,Z1]|M],A1).
% Funciones Auxiliares
moverC([X,Y,N],[X,Y1,N1],T):-N1 is N+1,Y1 is Y+1,Y1=<T.
moverC([X,Y,N],[X1,Y,N1],T):-N1 is N+1,X1 is X+1,X1=<T.
moverC([X,Y,N],[X,Y1,N1],_T):-N1 is N+1,Y1 is Y-1,Y1>0.
moverC([X,Y,N],[X1,Y,N1],_T):-N1 is N+1,X1 is X-1,X1>0.
verificaC(_C,[]):-!.
verificaC(C,[M|Ms]):-pertenece(M,C),verificaC(C,Ms).
% Tablero
celdaC(1,1).
celdaC(1,2).
celdaC(1,3).
celdaC(2,1).
celdaC(2,2).
```

celdaC(2,3).

celdaC(3,1).

celdaC(3,2).

celdaC(3,3).

% TAREAS - Examen Final del 28/07/03

% Debe armarse una lista de tareas (que deben hacerse en un cierto orden) que

% represente el menor costo.

% Datos

tarea(nombre(nil),precede(t1)).

tarea(nombre(t1),precede(t2)).

tarea(nombre(t1),precede(t3)).

tarea(nombre(t2),precede(t4)).

tarea(nombre(t3),precede(t4)).

tarea(nombre(t4),precede(nil)).

3

recursoTarea(t1,100,r1).

recursoTarea(t2,10,r1).

recursoTarea(t3,5,r2).

recursoTarea(t4,120,r1).

recurso(r1,'informatico').

recurso(r2,'humano').

% Predicados

% Ejemplo: secuenciaTareas(L).

% L=[nodoS(t1,informatico,100),nodoS(t3,humano,5),nodoS(t4,informatico,120)]

secuenciaTareas(L):-armarLdeL3(M,[]),listaMenorCosto(M,L).

armarLdeL3([X|L],Aux):-armarLista3(X,nil),not(pertenece(X,Aux)),!,

```
concatenar(Aux,[X],Aux2),armarLdeL3(L,Aux2).

armarLdeL3([],_Aux).

armarLista3([],E):-tarea(nombre(E),precede(nil)).

armarLista3([nodoS(T,D,C)|S],E):-tarea(nombre(E),precede(T)),

recursoTarea(T,C,I),recurso(I,D),

armarLista3(S,T).

listaMenorCosto([M|Ms],R):-sumaCosto(M,Cr),menorCostoAux(Ms,R,Cr,M),!.

menorCostoAux([],_M,_Cr,_M):-!.

menorCostoAux([M|Ms],R,Cr,_N):-sumaCosto(M,C),C<Cr,menorCostoAux(Ms,R,C,M).

menorCostoAux([_M|Ms],R,Cr,N):-menorCostoAux(Ms,R,Cr,N).

sumaCosto([nodoS(_,_,C)],C).

sumaCosto([nodoS(_,_,C)|S],C1):-sumaCosto(S,C2),C1 is C + C2.
```

% JULIO CESAR - Examen final del 29/07/02

```
%

% Si hay piratas en el camino, se resta 1 tripulante y se suman 2 en cada isla alcanzada.

%

% ?- julioCesar(L,30,X).

% L = [15, 16, 12, 11, 17, 14, 13].

% X = 39

%

% L = [15, 16, 14, 13].

% X = 33

%

% L = [15, 16, 17, 14, 13].

% X = 36

% DATOS %
```

```
isla(11).

isla(12).

isla(13).

isla(14).

isla(15).

isla(16).

isla(17).

% ruta(extremo1,extremo2,nroPiratas)

ruta(11,12,32).

ruta(12,16,0).

ruta(11,17,0).

ruta(15,16,0).

ruta(13,14,0).

ruta(17,14,0).

ruta(16,14,25).

ruta(16,17,0).
```



```
% camino(extremo1,extremo2,nroPiratas) los caminos son bidireccionales,

% el camino 11-12 es el mismo que el 12-11.

caminoJC(C1,C2,P):-ruta(C1,C2,P).

caminoJC(C1,C2,P):-ruta(C2,C1,P).

% Ubicación de la Sirena

sirena(13).

% Ubicación del Campitan

capitan(15).

% PROGRAMA %
```

```
% julioCesar(islas,tripInic,tripFinal)
julioCesar(L,N,X1):-capitan(C),julioAuxJC(L,N,X,C,[]),X1 is X-2.
% julioAux(islas,tripInic,tripFinal,capitan,visitadas)
julioAuxJC([L1,L2],N,X,L1,_T):-puedeAvanzarJC(L1,L2,N,X),sirena(L2).
julioAuxJC([L1|Ls],N,X,L1,T):-isla(L2),not(pertenece(L2,T)),
puedeAvanzarJC(L1,L2,N,X1),
concatenar([L2],T,Ti),julioAuxJC(Ls,X1,X,L2,Ti).
% puedeAvanzar(origen,destino,tripInic,tripFinal)
puedeAvanzarJC(L1,L2,N,X):-caminoJC(L1,L2,0),X is N+2,!.
puedeAvanzarJC(L1,L2,N,X):-caminoJC(L1,L2,P),P < N,X is N + 1.
```

% CAZADOR - Examen final del 03/05/05

```
% Para llegar a su presa, el cazador debe desplazarse diagonalmente por el tablero.
% Se especifica el tamano del tablero (N), posicion del Cazador (C) y de la presa (P).
%
% Ejemplo: cazador(3,[1,2],[1,2]). Yes
%
% Nota:
% Si la respuesta es NO, se sale fuera de la pila antes de responder.
% Si necesita mas pasos también se sale de la pila. ERROR: Out of global stack
cazador(N,C,P):-arribaDerecha(N,C,C1),abajoDerecha(N,C1,C2),
abajoIzquierda(N,C2,C3),arribaIzquierda(N,C3,P).
arribaDerecha(N,[X,Y],[X1,Y1]):-X1 is X +1, Y1 is Y+1, X1 =<N, Y1 =<N.
arribaDerecha(N,[X,Y],[X2,Y2]):-
arribaDerecha(N,[X,Y],[X1,Y1]),arribaDerecha(X,[X1,Y1],[X2,Y2]).
abajoDerecha(N,[X,Y],[X1,Y1]):-X1 is X +1, Y1 is Y-1, X1 =<N, Y1 >=1.
abajoDerecha(N,[X,Y],[X2,Y2]):-
```

```
abajoDerecha(N,[X,Y],[X1,Y1]),abajoDerecha(X,[X1,Y1],[X2,Y2]).

abajoIzquierda(_N,[X,Y],[X1,Y1]):-X1 is X -1, Y1 is Y-1, X1 >=1, Y1 >=1.

abajoIzquierda(N,[X,Y],[X2,Y2]):-

abajoIzquierda(N,[X,Y],[X1,Y1]),abajoIzquierda(X,[X1,Y1],[X2,Y2]).

arribaIzquierda(N,[X,Y],[X1,Y1]):-X1 is X -1, Y1 is Y+1, X1 >=1, Y1 =<N.

arribaIzquierda(N,[X,Y],[X2,Y2]):-

arribaIzquierda(N,[X,Y],[X1,Y1]),arribaIzquierda(X,[X1,Y1],[X2,Y2]).
```

5

% ARMAR LA DIETA - Examen 05/08/2002

```
%

% Se trata de armar la dieta por dia segun la cantidad de horas de estudio.

% Se informa una lista de lista cuyos elementos son [dia,horas], se

% devuelven las posibles dietas que pueden hacerse sumando las calorias y de

% acuerdo a las restricciones de calorias por horas de estudio (horasCalorias).

%

% Ejemplo: armarDieta([[lunes,2],[martes,4],[miercoles,3],[jueves,2],[viernes,3]],D).

% una de tantas soluciones:

% D=[[mantecaSancor],[pepsi],[mantecaSancor],[mantecaSancor],[mantecaSancor]]

producto(manteca,mantecaSancor,150).

producto(agua,nestle,60).

producto(manteca,mantecaLaSerenisima,200).

producto(leche,lecheSindor,190).

producto(margarina,danica,200).

producto(gaseosa,pepsi,300).

producto(leche,lecheLaCabania,102).

producto(leche,lecheCotapa,90).
```

```
producto(carne,vaca,200).
producto(mayonesa,hellman,190).
producto(huevos,cordoniz,140).
producto(gaseosa,frescor,140).
producto(pan,centeno,100).
producto(pan,lactal,190).
producto(pan,harina,120).
producto(leche,lechePolvo,110).
producto(arroz,arrozSanSalvador,110).
producto(mayonesa,mayoDanica,180).
producto(manteca,fredo,170).
producto(agua,aguaSanSalvador,70).
producto(gaseosa,cocaCola,120).
producto(gaseosa,sevenUp,150).
horasCalorias(1,50).
horasCalorias(2,100).
horasCalorias(3,100).
horasCalorias(4,200).
horasCalorias(5,200).
horasCalorias(6,300).
horasCalorias(7,300).
horasCalorias(8,400).
armarDieta([],[]).
armarDieta([[_X1,X2]|Xs],[Y|Ys]):-horasCalorias(X2,D),dieta(Y,[],0,D),
armarDieta(Xs,Ys).
dieta([Y],AUX,CAL,CM):-producto(T,Y,C),
```

```
ocurre(producto(T,_,_),AUX,R),R<4,
ocurre(producto(T,Y,C),AUX,M),M<3, N is C + CAL,CM < N.
dieta([Y|Yr],AUX,CAL,CM):-producto(T,Y,C),
ocurre(producto(T,_,_),AUX,R),R<4,
ocurre(producto(T,Y,C),AUX,M),M<3,
concatenar(AUX,[producto(T,Y,C)],AUX2),N is C + CAL,
CM > N, dieta(Yr,AUX2,N,CM).
```

% MUJER FIEL - Examen final

```
%
% Dado una serie de predicados, se trata de averiguar si una mujer ha sido fiel.
pareja(benito,andrea,[03,10,2001],[10,08,2002]).
pareja(carlos,maria-jose,[01,05,2001],[02,05,2001]).
pareja(santiago,andrea,[02,02,1998],[10,08,1998]).
pareja(santiago,stella,[06,03,1998],[12,06,1998]).
pareja(juan,maria-jose,[03,07,2000],[10,08,2002]).
pareja(pepe,andrea,[03,10,2001],[10,08,2002]).
6
% formato [01,02,1998]
fma([_A,_B,C],[_X,_Y,Z]):-C>Z,!.
fma([_A,B,C],[_X,Y,Z]) :-C=:=Z,B>Y,!.
fma([A,B,C],[X,Y,Z]) :-C=:=Z,B=:=Y,A>=X,!.
% el segundo par de fechas es posterior al primero
superpuestas(A,B,C,_D):-fma(C,A),fma(B,C).
mujerInfiel(M):-pareja(V1,M,Fi1,Ff1),pareja(V2,M,Fi2,Ff2),not(V1=V2),
superpuestas(Fi1,Ff1,Fi2,Ff2).
mujerFiel(M):-not(mujerInfiel(M)).
```

% EJERCICIO DE EXAMEN - JUEGO 4 EN LÍNEA

```
%jugada(Tablero,ColorFicha,NroColumna,TableroResultado).

jugada(T,F,N,T):-rotarH(T,Tr),enesimoElem(Tr,N,C),not(pertenece(x,C)),!.

jugada(T,F,N,R):-rotarH(T,Tr),enesimoElem(Tr,N,C),xPosicion(x,C,P),

insertarXenN(C,P,F,C1),insertarXenN(Tr,N,C1,Tra),rotarA(Tra,R).

%?- jugada([[v,x,v,v,v,n],

% [v,x,v,v,v,n],

% [v,v,v,v,v,n],

% [v,v,v,v,v,n],

% [v,v,v,v,v,n],

% [v,v,v,v,v,n],

% [v,v,v,v,v,n]],n,2,R).

%R = [[v,x,v,v,v,n],

% [v,n,v,v,v,n],

% [v,v,v,v,v,n],

% [v,v,v,v,v,n],

% [v,v,v,v,v,n],

% [v,v,v,v,v,n],

% [v,v,v,v,v,n]]

% cuatroEnLinea(Tablero,Color). Responde yes si hay cuatro

% fichas en linea de color Color.

cuatroEnLinea(T,C):-checkHorizontal(T,C),!.

cuatroEnLinea(T,C):-checkVertical(T,C),!.

cuatroEnLinea(T,C):-checkDiagonal(T,C),!.

checkHorizontal([F|Fs],C):-subLista([C,C,C,C],F),!.

checkHorizontal([F|Fs],C):-checkHorizontal(Fs,C).
```

```
checkVertical(T,C):-rotarH(T,Tr),checkHorizontal(Tr,C).

checkDiagonal(T,C):-obtenerDiagonal(T,D),subLista([C,C,C,C],D).

obtenerDiagonal(M,D):-aux4l(N),diagonalN(M,N,D).

obtenerDiagonal(M,D):-rotarA(M,Mr),aux4l(N),diagonalN(Mr,N,D).

obtenerDiagonal(M,D):-rotarA(M,Mr1),rotarA(Mr1,Mr2),aux4l(N),diagonalN(Mr2,N,D).

obtenerDiagonal(M,D):-rotarH(M,Mr),aux4l(N),diagonalN(Mr,N,D).

aux4l(1).

aux4l(2).

aux4l(3).

%Ejemplos.. aceptamos que las piezas floten :)

% cuatroEnLinea([[x,v,x,x,x,x],[x,x,v,x,x,x],[x,x,x,v,x,x],[x,x,x,x,v,x],[x,x,x,x,x,x],[x,v,v,x,v,x],
[x,n,n,v,n,x]],v).

%

% Yes

% cuatroEnLinea([[x,v,x,x,x,x],[x,x,v,x,x,n],[x,x,x,v,n,x],[x,x,x,n,v,x],[x,x,n,x,x,x],[x,v,v,x,v,x],
[x,n,n,v,n,x]],n).

% Yes
```

Factorial:

```
factorial(X,Y):-X<1,!,fail.

factorial(1,1):-!.

factorial(X,Y):- X1 is X-1, factorial(X1,Y2),Y is X*Y2.
```

Aplanar:

```
aplanar([],[]):-!.

aplanar([X|C],L):- aplanar(X,L1), aplanar(C,L2), concatenar(L1,L2,L), !.

aplanar(X,[X]).
```

Aplanar de a un nivel:

```
aplana1([],[]):-!.

aplana1(X,[X]):- atomic(X),!.

aplana1([X|Y],L):-  aplanar([X],T), T=[X], aplana1(Y,L1), append([X],L1,L),!.

aplana1([X|Y],L):-  aplana1(Y,L1), append(X,L1,L),!.
```

Menor:

```
menor(X,Y,Y):-X>Y,!.

menor(X,Y,X).

menor([X|[]],X):-!.

menor([X|C],M):- menor(C,M1),menor(X,M1,M).
```

Borrar:

```
borrar([],M,[]).

borrar([X|C],M,C):- X==M,!.

borrar([X|C],M,L):- borrar(C,M,L2), concatenar([X],L2,L).
```

Ordenar:

```
ordenar([],[]):- !.

ordenar(X,[M|C]):- menor(X,M), borrar(X,M,L1), ordenar(L1,C).
```

Ordenar y aplanar:

```
aplaord(L,LO):- aplanar(L,L1), isnumero(L1), ordenar(L1,LO).
```

Validar si es numero:

```
isnumero([L|[]]):-integer(L), !.

isnumero([L|C]):-integer(L), isnumero(C).
```

TOTAL de permutaciones de N elementos:

```
extract(X,[X|Xs],Xs).

extract(X,[Y|Ys],[Y|Zs]):- extract(X,Ys,Zs).

permutaciones([ ], [ ]).

permutaciones(X, [Z|Z1]):- extract(Z, X, Y), permutaciones(Y,Z1).
```

TOTAL de permutaciones de TODOS los elementos que suman un determinado numero (made in Pablito):

```
probar2([], N, []).

probar2([X|C], N, L):- N>=X, N1 is N-X, probar(C, N1, L1), append([X], L1, L).

probar2([X|C], N, L):- probar(C, N, L).
```

```
probar([], 0, []).

probar([X|C], N, L):- N>=X, N1 is N-X, probar(C, N1, L1), append([X], L1, L).

probar([X|C], N, L):- probar(C, N, L).
```

Longitud de una lista:

```
largo([],0).

largo([X|C],H):- largo(C,H1), H is H1+1.
```

Concatenar:

```
concatenar([], L, L).

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

Semejanza entre dos palabras (suma 1 si es igual, resta 1 si es distinto):

```
semejanza([],C,S):- largo(C,H),S is 0-H,!.

semejanza([X|C],[],S):- largo(C,H),S is -H-1,!.

semejanza([X|C1],[X|C2],S):- semejanza(C1,C2,S1), S is S1+1,!.

semejanza([_|C1],[_|C2],S):- semejanza(C1,C2,S1), S is S1-1.
```

Buscar aplicable a "Semejanza":

```
buscar(X,[],[]).
```

```
buscar(X,[L|C],S):-atom_chars(X,L1), atom_chars(L, L2), semejanza(L1,L2,H), H>0, buscar(X,C,S2), concatenar([L],[H],A), concatenar([A],S2,S),!.

buscar(X,[L|C],S):-atom_chars(X,L1), atom_chars(L, L2), semejanza(L1,L2,H), H=<0, buscar(X,C,S),!.


buscarigual(X,[_|C]):-buscarigual(X,C), !.

buscarigual(X,[X|C]):- !.


buscariguales(X,L):-dic(S), not(buscarigual(X,S)), buscar(X,S,L),!.

buscariguales(X,X).
```

Reemplazo especial (TP 11, 6 parametros):

```
valpos(X, Y, S):- length(Y,S), X>=1, X=<S, !.

valpos(X, Y, S):- display('Posicion no valida'), fail, !.


valcant(Cant, Pos, S):- (Cant + Pos)=<S, !.

valcant(Cant, Pos, S):- display('El parametro 5 ha fallado'), fail, !.


acorta([D|C],Pos,L, []):- Pos=<1, append([],[D|C],L), !.

acorta([D|C],Pos,L, [D|I]):- X1 is Pos-1, acorta(C,X1, L, I), !.


intercambio([],Elem, Cant, X, []):- !.

intercambio([Elem|T], Elem, Cant, X, LF):- Cant>0, E is Cant-1, intercambio(T,Elem,E,X, S), append([X],S,LF), !.

intercambio([Y|T],Elem, Cant, X, LF):-  Cant>0, intercambio(T,Elem,Cant,X, S), append([Y],S,LF), !.

intercambio([Y|T], Elem, Cant, X, LF):- Cant=0, intercambio(T,Elem,Cant,X, S), append([Y],S,LF).
```

```
reemplazar(L,Elem,X,Pos,Cant,R):- Cant=(-1), valpos(Pos, L, S), valcant(Cant,Pos, S), acorta(L,Pos, T, I), intercambio(T, Elem, S, X, P), append(I, P, R), !.

reemplazar(L,Elem,X,Pos,Cant,R):- valpos(Pos, L, S), valcant(Cant,Pos, S), acorta(L,Pos, T, I), intercambio(T, Elem, Cant, X, P), append(I, P, R), !.

reemplazar(L,Elem,X,Pos,Cant,R):- display('fallo'), fail.
```

CONJUNTO:

a-

```
pertenece_conjunto(X, [X|_]).

pertenece_conjunto(X, [_|L]) :- pertenece_conjunto(X, L).

es_conjunto([]).

es_conjunto([X|C]) :- not(pertenece_conjunto(X, C)), es_conjunto(C).
```

b-

```
pertenece_conjunto(X, [X|_]).

pertenece_conjunto(X, [_|L]) :- pertenece_conjunto(X, L).
```

c-

```
insertar(X, L, [X|L]) :- not(pertenece_conjunto(X, L)).
```

d-

```
union([], L, L).

union([X1|C1], C2, [X1|R2]) :- es_conjunto([X1|C1]), es_conjunto(C2), not(pertenece_conjunto(X1, C2)), union(C1, C2, R2).

union([X1|C1], C2, R2) :- es_conjunto([X1|C1]), es_conjunto(C2), pertenece_conjunto(X1, C2), union(C1, C2, R2).
```

e-

```
interseccion([], _, []).

interseccion([X1|C1], C2, [X1|R]) :- es_conjunto([X1|C1]), es_conjunto(C2), pertenece_conjunto(X1, C2), interseccion(C1, C2, R).

interseccion([X1|C1], C2, R) :- es_conjunto([X1|C1]), es_conjunto(C2), not(pertenece_conjunto(X1, C2)), interseccion(C1, C2, R).
```

f-

```
diferencia([], _, []).

diferencia([X1|C1], C2, [X1|R]) :- es_conjunto([X1|C1]), es_conjunto(C2), not(pertenece_conjunto(X1, C2)), diferencia(C1, C2, R).

diferencia([X1|C1], C2, R) :- es_conjunto([X1|C1]), es_conjunto(C2), pertenece_conjunto(X1, C2), diferencia(C1, C2, R).
```

g-

```
construir_conjunto([], []).

construir_conjunto([X|C], [X|R]) :- construir_conjunto(C, R), not(pertenece_conjunto(X, R)).
```

```
construir_conjunto([X|C], R) :- construir_conjunto(C, R), pertenece_conjunto(X, R).
```

GRAFO TP 12:

```
arista(a,b).
arista(a,c).
arista(a,e).
arista(a,d).
arista(b,c).
arista(c,d).
arista(c,e).
arista(d,e).

conectado(X,Y) :-
   arista(X,Y); arista(Y,X).

aristas(L) :-
   findall([X,Y], conectado(X,Y),L).

nodos(L) :-
   findall(X,conectado(X,_), L1),
   findall(Y,conectado(_,Y),L2),
   append(L1,L2,L3),
   sort(L3,L).

borrar(_,[],[]) :- !.
```

```
borrar([X,Y], [[X1,Y1]|R],R1) :-
   (X = X1 , Y = Y1;  Y =  X1, X = Y1), !,
   borrar([X,Y],R,R1).
borrar([X,Y], [[X1,Y1]|R],[[X1,Y1]|R1]) :-
   borrar([X,Y],R,R1).

recorrido(R) :-
   aristas(La),
   nodos(Ln),
   member(X,Ln),
   camino(X,La,R).

camino(X,[],[X]) :- !.
camino(X,L1,[X|R]) :-
   conectado(X,Y),
   member([X,Y],L1),
   borrar([X,Y],L1,L),
   camino(Y,L,R).
```

LABERINTO TP 12:

```
arista(inicio, 1).
arista(1, 2).
arista(1, 3).
arista(2, 4).
arista(3, 4).
```

```
arista(4, fin).

arista_b(X, Y):- arista(X, Y); arista(Y, X).

camino(fin, L, L):- !.
camino(X, L, Lr):- arista_b(X, Y), not(member(Y, L)), camino(Y, [Y|L], Lr).

recorrer(Lr):- camino(inicio, [inicio], L), reverse(L, Lr).
```

Invertir:

```
inv([],[]).
inv([X|Y],L):- inv(Y,L1), conc(L1,[X],L).
```

Cuenta Positivos:

```
cpos([],0).
cpos([K|X],C):- K@>=0 -> cpos(X,C1) -> C is C1+1,!.
cpos([K|X],C):-  cpos(X,C1) -> C is C1.
```

Separa positivos y negativos en dos listas diferentes:

```
sep([],[],[]).
sep([X|Y],P,N):- X@>=0 -> sep(Y,P1,N),conc([X],P1,P),!.
sep([X|Y],P,N):- sep(Y,P,N1),conc([X],N1,N).
```

Maximo y minimo:

```
ma([X], X) :- !.
ma([X1, X2 | L], X) :- X1 @>= X2 -> ma([X1|L], X); ma([X2|L], X).

mi([X], X) :- !.
mi([X1, X2 | L], X) :- X1 @=< X2 -> mi([X1|L], X); mi([X2|L], X).

mm([],0,0).
mm(X,MA,MI):- ma(X,MA), mi(X,MI).
```

Ejercicio Mapa de la Casa: (agregar para ue sea bidireccional)

```
conectado(A,B):- A=a,B=b,!.
conectado(A,D):- A=a,D=d,!.
conectado(B,C):- C=c,B=b,!.
conectado(C,G):- G=g,C=c,!.
conectado(D,F):- D=d,F=f,!.
conectado(E,F):- E=e,F=f,!.
conectado(F,G):- F=f,G=g,!.
conectado(G,S):- G=g,S=salida.

salid(S1,[]):-  S1=salida,!.
salid(X,[Y|C]):- conectado(X,Y) , salid(Y,C).
```

Extrae numeros de la lista:

```
snum([],[]).

snum([X|XC],Y):- number(X), snum(XC,Y1), conc([X],Y1,Y), !; snum(XC,Y1), conc([],Y1,Y).
```

Obtener ciclo de euler (Made in QK):

```
con(A,B):- A=a , B=b,!; A=b, B=a,!.

con(B,C):- B=b , C=c,!; B=c, C=b,!.

con(C,D):- C=c , D=d,!; C=d, D=c,!.

con(E,D):- E=e , D=d,!; E=d, D=e,!.

con(E,H):- E=e , H=h,!; E=h, H=e,!.

con(E,G):- E=e , G=g,!; E=g, G=e,!.

con(E,C):- E=e , C=c,!; C=e, E=c,!.

con(G,H):- G=g , H=h,!; G=h, H=g,!.

con(F,G):- F=f , G=g,!; F=g, G=f,!.

con(A,C):- A=a , C=c,!; C=a, A=c,!.

con(A,F):- A=a , F=f,!; A=f, F=a,!.

con(C,F):- C=c , F=f,!; C=f, F=c,!.

con(C,G):- C=c , G=g,!; C=g, G=c,!.

elimino([X|_],[],[]):- !.

elimino([X|_],[X|L],L):- !.

elimino([X|_],[Y|L],L1):- elimino([X],L,L2),conc([Y],L2,L1).

aux(A,[],[]):- !.

aux(A,[X|L1],C):- con(A,X) -> conc([X],[],C),!; aux(A,L1,C).
```

```
ciclo([A|_],[],[]):- !.

ciclo([A|_],[X|L], C):- aux(A,[X|L],Y), elimino(Y,[X|L],Z), ciclo(Y,Z,Y1), conc(Y,Y1,C),!.
```

Reinas(funcion madre: validar), devuelve lista de reinas q me atacan en la coordenada que le paso yo como parametro!:

```
reinas([[1,1],[3,4],[5,7],[8,8],[6,2]]).

diagonal_1(X,_,_,[]):- X@>8, !.

diagonal_1(_,Y,_,[]):- Y@>8, !.

diagonal_1(X,Y,R,L):- X1 is (X+1),
                      Y1 is (Y+1),
                      pertenec([X,Y],R),
                      diagonal_1(X1,Y1,R,L1),
                      conc([[X,Y]],L1,L),!;
                      X1 is (X+1),
                      Y1 is (Y+1),
                      diagonal_1(X1,Y1,R,L).

diagonal_2(X,_,_,[]):- X@<0, !.

diagonal_2(_,Y,_,[]):- Y@<0, !.

diagonal_2(X,Y,R,L):- X1 is (X-1),
                      Y1 is (Y-1),
                      pertenec([X,Y],R),
```

```
                diagonal_2(X1,Y1,R,L1),
                conc([[X,Y]],L1,L),!;
                X1 is (X-1),
                Y1 is (Y-1),
                diagonal_2(X1,Y1,R,L).

diagonal_3(X,_,_,[]):- X@<0, !.
diagonal_3(_,Y,_,[]):- Y@>8, !.
diagonal_3(X,Y,R,L):- X1 is (X-1),
                Y1 is (Y+1),
                pertenec([X,Y],R),
                diagonal_3(X1,Y1,R,L1),
                conc([[X,Y]],L1,L),!;
                X1 is (X-1),
                Y1 is (Y+1),
                diagonal_3(X1,Y1,R,L).

diagonal_4(X,_,_,[]):- X@>8, !.
diagonal_4(_,Y,_,[]):- Y@<0, !.
diagonal_4(X,Y,R,L):- X1 is (X+1),
                Y1 is (Y-1),
                pertenec([X,Y],R),
                diagonal_4(X1,Y1,R,L1),
                conc([[X,Y]],L1,L),!;
                X1 is (X+1),
                Y1 is (Y-1),
```

```
                diagonal_4(X1,Y1,R,L).


vertical(X,_,_,[]):- X@>8 ,!.
vertical(X,Y,R,L):-    X1 is X+1,
                pertenec([X,Y],R),
                vertical(X1,Y,R,L1),
                conc([[X,Y]],L1,L),!;
                X1 is (X+1),
                vertical(X1,Y,R,L).


lateral(_,Y,_,[]):- Y@>8 ,!.
lateral(X,Y,R,L):-   Y1 is Y+1,
                pertenec([X,Y],R),
                lateral(X,Y1,R,L1),
                conc([[X,Y]],L1,L),!;
                Y1 is (Y+1),
                lateral(X,Y1,R,L).


validar(X,_,_):-  X@>8 -> write('coordena invalida'), !;
              X@<0-> write('coordena invalida'),!.
validar(_,Y,_):-  Y@>8 -> write('coordena invalida'), !;
              Y@<0-> write('coordena invalida'),!.
validar(X,Y,_):- reinas(R), pertenec([X,Y],R) -> write('coordenada ocupada'),!.
validar(X,Y,L):- reinas(R),
             diagonal_1(X,Y,R,L1),
             diagonal_2(X,Y,R,L2),
```

```
        diagonal_3(X,Y,R,L3),
        diagonal_4(X,Y,R,L4),
        vertical(1,Y,R,L5),
        lateral(X,1,R,L6),
        conc(L1,L2,LL1),
        conc(L3,L4,LL2),
        conc(L5,L6,LL3),
        conc(LL1,LL2,LLL1),
        conc(LL3,LLL1,L).
```

Cuenta los seguidos iguales (nomas los primeros):

```
cue_iw(X,[],0):- !.
cue_iw(X,[X|L],C):- cue_iw(X,L,C1), C is C1+1,!.
cue_iw(X,[Y|L],0).
```

```
cuenta_iw([X],1):- !.
cuenta_iw([X|L], C):- cue_iw(X,[X|L],C).
```

Corta N elementos:

```
corta([],_,[]):- !.
corta(L,0,L):- !.
corta([X|L],N,Lf):- N1 is N-1, corta(L,N1,Lf).
```

Arma lista con cantidades de elementos iguales:

```
repet([],[]):- !.

repet(L,Lf):- cuenta_iw(L,C), corta(L,C,LL), repet(LL, L1), conc([C],L1,Lf).
```

FIBONACCI:

```
fibonacci(0,0):- !.

fibonacci(1,1):- !.

fibonacci(N,C):- N1 is N-1,N2 is N-2,fibonacci(N1,C1), fibonacci(N2,C2), C is C1+C2.
```

Agregar un elemento en una posicion:

```
agregar_elem([],X,_,[X]):- !.

agregar_elem(L,X,C,[X|L]):- C@=<1, !.

agregar_elem([Y|L],X,C,Lf):- C1 is C-1, agregar_elem(L,X,C1,La), conc([Y],La,Lf).
```

Longitud par o impar:

```
parimpar([],par):- !.

parimpar(L,X):- long(L,S), R is S mod 2, R=0 -> X=par; X=impar.
```

PAR:

```
par(X):- X1 is X mod 2, X1=0.
```

Sublista a partir de un elemento (no posicion) con una cantidad dada de elementos:

```
sub(_,0,[]):- !.

sub([X|L],C,SL):- C1 is C-1, sub(L,C1,S), conc([X],S,SL).


sublista([],_,_,[]):- !.

sublista([X|L],X,C,SL):- sub([X|L],C,SL),!.

sublista([X|L],Y,C,SL):- sublista(L,Y,C,SL).
```

Ejercicio 1, 1er parcial Scheme (ZIPSETLIST):

```
cuentaconsecutivo([X|[]], 1):- !.

cuentaconsecutivo([X, X|C], N):- cuentaconsecutivo([X|C], N1), N is N1+1, !.

cuentaconsecutivo([X, Y|C], 1).

cuentaconsecutivo([X, Y|C], N):- cuentaconsecutivo([Y|C], N1), N is 1.


cortar([X|C], 0, [X|C]):- !.

cortar([], Cant, []):- !.

cortar([X|C], Cant, LC):- Cant1 is Cant-1, cortar(C, Cant1, LC).


zipsetlist([], []):- !.

zipsetlist([X|C], L):- cuentaconsecutivo([X|C], N), cortar([X|C], N, LL), zipsetlist(LL, P), append([X],
[N], Q), append([Q], P, L).
```

Ejercicio 2, 1er parcial Scheme (SUMA RUSA):

```
par(0):- !.

par(X):- X>0, X1 is X-2, par(X1).


formarlistas(0, Y, []):- !.

formarlistas(X, 0, []):- !.

formarlistas(X, Y, L):- X1 is X//2, Y1 is Y*2, formarlistas(X1, Y1, LL), append([X], [Y], P), append([P], LL, L).


suma([], 0):- !.

suma([[X|Y]|C], Z):- par(X)->suma(C,Z);suma(C,Z1), Z is Z1+Y.


sumarusa(X, Y, Z):- formarlistas(X, Y, L), suma(L, Z).
```

Concatena especial (Intercala elementos, andres hacia problemas):

```
sublista([], Cant, []):- !.

sublista([X|C], 0, []):- !.

sublista([X|C], Cant, L):- Cant1 is Cant-1, sublista(C, Cant1, LL), append([X], LL, L).


cortar([X|C], 0, [X|C]):- !.

cortar([], Cant, []):- !.

cortar([X|C], Cant, LC):- Cant1 is Cant-1, cortar(C, Cant1, LC).


concatenaespecial(_, L2,  0, _, L2):- !.

concatenaespecial(L1, _,  _, 0, L1):- !.

concatenaespecial(_, _,  0, 0, []):- !.
```

```
concatenaespecial(L1, [],  _, _, L1):- !.

concatenaespecial([], L2,  _, _, L2):- !.

concatenaespecial(X1, X2, Cant1, Cant2, L):- sublista(X1, Cant1, L1), sublista(X2, Cant2, L2),
cortar(X1, Cant1, LL1), cortar(X2, Cant2, LL2), concatenaespecial(LL1, LL2, Cant1, Cant2, LLL),
append(L1, L2, L3), append(L3, LLL, L),!.
```

Preparcial 2009

OPERAR:

```
operar(X, Y, '+', R):- R is X+Y, !.

operar(X, Y, '-', R):- R is X-Y, !.

operar(X, Y, '*', R):- R is X*Y, !.

operar(X, Y, '/', R):- (Y=0)->display('Division por cero!'), !;R is X/Y, !.


opera(X, Y, S, R):- operar(X, Y, S, R).
```

PERMUTACION RARA:

```
probar([], 0, []).

probar([X|C], N, L):- N>=X, N1 is N-X, probar(C, N1, L1), append([X], L1, L).

probar([X|C], N, L):- probar(C, N, L).
```

GRAFO (MADE IN QK):

```
arista(a, b, 5).
```

```
arista(b, c, 7).

arista(c, d, 3).

arista(d, e, 3).

arista(e, a, 4).


unida(X,Y,P):- arista(X,Y,P1), P is P1,!; arista(Y,X,P1), P is P1.


%camino(X,Y,C,P):- unida(X,Y1,P1), Y1=Y, P is P1, append([[X,Y]],[],C),!.

%camino(X,Y,C,P):- unida(X,Y1,P1), camino(Y1,Y,C1,P2), P is P1+P2, append([[Y1,Y]],C1,C).


cam(X,F,[],[],0):- !.

cam(_,F,L,_,_):- not(pertenec(F,L)),!.

cam(X,F,[F|L],C,P):- unida(X,F,P), conc([[X,F]],[],C),!.

cam(X,F,[Y|L],C,P):- unida(X,Y,P1), cam(Y,F,L,C1,P2), P is P1+P2, conc([[X,Y]],C1,C),!; cam(X,F,L,C,P).


caminos(X,F,L,C,P):- per(L,L1), cam(X,F,L1,C,P).
```

Pre-parcial 2010

```
%ejercicio 7 (Quitar duplicados)


pert(X, []):- fail,!.

pert(X,[X|C]):- !.

pert(X,[_|C]):- pert(X,C).


quitar([],[]):- !.
```

```
quitar([X|C],L):- not(pert(X,C)), quitar(C,L1), conc([X],L1,L),!.

quitar([X|C],L):- quitar(C,L).


%-------------------------------------------------------------------------------

%ejercicio 8 (Suma posiciones impares y suma posiciones pares)


sumar([],0,0):- !.

sumar([X],X,0):- !.

sumar([X,Y|C],SI,SP):- sumar(C,SI1,SP2), SI is SI1+X, SP is SP2+Y.


%-------------------------------------------------------------------------------

%ejercicio 4 (Pasar de una forma a otra (GRAFOS))

%[[a, [b]], [b, [c, d]], [c, [e]], [d, [a, e]], [e, [a]]]

%[[a, b, c, d, e], [[a, b], [b, c], [b, d], [c, e], [d, a], [d, e], [e, a]]]


obtener_nodos([],[]):- !.

obtener_nodos([[X|_]|C],L):- obtener_nodos(C,L1), conc([X],L1,L).


apl([], []):- !.

apl([X|L], L2) :- X = [_|_], !, apl(X, X1), apl(L, L1), append(X1, L1, L2),!.

apl([X|L], [X|L1]) :-  apl(L, L1).


aristas_aux([X|[]],[]):- !.

aristas_aux([X,Y|C],A):- aristas_aux([X|C],A1), conc([[X,Y]],A1,A).


obtener_aristas([],[]):- !.
```

```
obtener_aristas([X|R],A):-  obtener_aristas(R,A1), apl(X,LL), aristas_aux(LL,L), conc(L,A1,A).

chupamelaa(L1,Lf):- obtener_nodos(L1,LN), obtener_aristas(L1,LA), conc([LN],[LA],Lf).
```

Invierte TODO!, se mete en las profundidades mas oscuras...:

```
conc([],J,J).
conc([X|X1],J,[X|L]):- conc(X1,J,L).

invierte([],[]):- !.
invierte([X|L],LL):- atomic(X), invierte(L,P), conc(P,[X],LL),!.
invierte([X|L],I):- invierte(X,V), invierte(L,I1), conc(I1,[V],I).
```

EXAMEN ARBOL (Made in QK):

```
arbol(a,[b,c,d],[3,2,4],[b,c,d,e,f]).
arbol(b,[],[],[]).
arbol(c,[e,f],[3,5],[e,f]).
arbol(d,[],[],[]).
arbol(e,[],[],[]).
arbol(f,[],[],[]).

buscapeso([X|L],[Y|K],X,Y):- !.
buscapeso([X|L],[Y|K],N,S):- buscapeso(L,K,N,S).

eshijo(R,N,S):- arbol(R,L,P,_), pertenec(N,L), buscapeso(L,P,N,S).
```

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

esdecen(R,N):- arbol(R,_,_,L), pertenec(N,L).


camino(R,R,[],0).

camino(R,N,_,_):- not(esdecen(R,N)), !.

camino(R,N,C,P):- eshijo(R,N,P1), P=P1, conc([[R,N]],[],C).

camino(R,N,C,P):- tienehijos(R), esdecen(R,N), arbol(R,[X|L],[Y|K],_), camino(X,N,C1,P1),
conc([R],C1,C).
```

Profundidad de una lista:

```
aplana([],[]).

aplana([X|Y],L):- es_lista(X), !, aplana(X,L1), aplana(Y,L2), append(L1,L2,L).

aplana([X|Y],[X|L]):-  aplana(Y,L).


aplana1([],[]):-!.

aplana1(X,[X]):- atomic(X),!.

aplana1([X|Y],L):-  aplana([X],T), T=[X], aplana1(Y,L1), append([X],L1,L),!.

aplana1([X|Y],L):-  aplana1(Y,L1), append(X,L1,L),!.


profund([],0):- !.

profund(L,V):- aplana(L,T), T=L,V is 1,!.

profund(L,V):- aplana(L,T), aplana1(L,P), T=P,V is 2,!; aplana1(L,P), profund(P,V1),V is V1+1.
```

Profundidad(Made in Pablo):

```
profund([], 1):- !.

profund(X, 0):- atomic(X), !.

profund([X|C], P):- is_list(X), profund(X, P1), P is P1+1, !.

profund([X|C], P):- atomic(X), profund(C, P), !.


profundidad([], 1):- !.

profundidad([X, Y|C], P):- profund(X, P1), profund(Y, P2), P1>P2, profundidad([X|C], P), !.

profundidad([X, Y|C], P):- profund(X, P1), profund(Y, P2), profundidad([Y|C], P), !.

profundidad([X], P):- profund(X, P1), P is P1+1, !.
```

Agrupar(si existe al lado, sino al final):

```
agrupar(L,X,L1):- not(pertenece(X,L)), append(L,[X],L1).

agrupar([X|L],X,L1):- append([X,X],L,L1),!.

agrupar([Y|L],X,L1):- agrupar(L,X,L2), append([Y],L2,L1),!.


% ********** VARIOS **********


/* suma de los elementos de una lista */

total([],0).

total([C|L],T):-

total(L,T1),

T is T1+C.


/* longitud de una lista */

lenght([],0).
```

```
lenght([_|L],T):-
lenght(L,T1),
T is T1+1.

/* adicionar un elemento de la cabeza de una lista */
addhead(X, L, [X|L]).

/* borrar la cabeza de una lista*/
deletehead(L,L1):-
addhead(_,L1,L).

/* adicionar al final de una lista */
addend(X, [], [X]).
addend(X, [C|R], [C|R1]):-
addend(X, R, R1).

/* borrar el ultimo elemento de una lista */
deleteend(L,L1):-
addend(_,L1,L).

/* borrar un elemento de una lista dado el indice */
delete(Indice,L,L1):-
insert(_,Indice,L1,L).

/* insertar un elemento en una lista dado el indice en que se quiere insertar*/
insert(X,0,L1,[X|L1]).
```

```
insert(X,Pos,[C|R],[C|R2]):-
Pos1 is Pos-1,
insert(X,Pos1,R,R2).

/* devuelve las posiciones en que se encuentra un elemento X*/
pos(X,[X|_],0).
pos(_,[],_):-
!,fail.
pos(X,[_|R],Pos):-
pos(X,R,Pos1),
Pos is Pos1+1.

/* clonar lista*/
clonlist([], []).
clonlist([C|R], [C|R1]):-
clonlist(R, R1).
/* elemento X de una Lista*/
getElem(0,[C|_],C):-!.
getElem(X,[_|R],Sol):-
X1 is X -1,
getElem(X1,R,Sol)
.

/* existencia de un elemento en una lista */
existe(_,[]):-fail.
existe(X,[X|_]):-!.
```

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

/* elminar un elemento de la lista */
eliminar(_,[],[]):-fail.
eliminar(X,[X|R],R).
eliminar(X,[C|R],[C|R1]):-
eliminar(X,R,R1)
.

/* subconjuntos de una lista */
subconjunto([],[]).
subconjunto([C|R],[C|R1]):-
subconjunto(R,R1).
subconjunto(L,[_|R1]):-
subconjunto(L,R1).

/* permutaciones de una lista*/
permutaciones([],[]).
permutaciones([C1|R1],L):-
eliminar(C1,L,Rest),
permutaciones(R1,Rest).

/* laboratorio */
vacia([]).
subcPerm(L,L1):-
```

```
subconjunto(T,L1),
permutaciones(L,T)
.

check([],[],A,A).
check([C|R],[+C|R1],SumaTemp,Suma):-
M is SumaTemp + C,
check(R,R1,M,Suma)

.
check([C|R],[-C|R1],SumaTemp,Suma):-
M is SumaTemp - C,
check(R,R1,M,Suma)
.

lab(ListaLarga,Suma,[C|R1]):-
subcPerm([C|R],ListaLarga),
check(R,R1,C,Suma)
.

/* invertir una lista*/
invertir([],[]).
invertir([C|R],L):-
invertir(R,X),
addend(C,X,L).
```

```
/* mayor de una lista */

mayor([C|[]],C):-!.

mayor([C|R],C1):-

mayor(R,C2),

C&gtC2 ,

C1 is C,!.

mayor([_|R],C1):-

mayor(R,C1)

.


/* menor de una lista */

menor([C|[]],C):-!.

menor([C|R],C1):-

menor(R,C2),

C&ltC2 ,

C1 is C,!.

menor([_|R],C1):-

menor(R,C1)

.


/* sublistas de una lista*/

prim([],_).

prim([C|R],[C|R1]):-

prim(R,R1)

.

sublista([],[]).
```

```
sublista([C|R],[C1|R1]):-
prim([C|R],[C1|R1]);
sublista([C|R],R1)
.

/* verifica si una lista es creciente*/
creciente([_|[]]).
creciente([C|[C1|R1]]):-
C < C1,
creciente([C1|R1])
.

/* calcula los intervalos crecientes de una lista */
intervalosCrec(Inter,L):-
sublista(Inter,L),
creciente(Inter)
.

may(Inter,L, Long):-
( intervalosCrec(Inter,L),lenght(inter,Long) );
( Long1 is Long -1,may(Inter,L,Long1) )
.

l(Inter,L):-
lenght(L,M);
may(Inter,L,M)
```

```
.
/* producto de 2 vectores */
prodEscalar([],[],0).
prodEscalar([C|R],[C1|R1],Result):-
prodEscalar(R,R1,Result1),
Result is C * C1 + Result1
.
/* cantidad columnas de una matriz */
cantCol([C|_],CC):-
lenght(C,CC)
.

/* cantidad filas de ina matriz */
cantFil(L,CF):-
lenght(L,CF)
.

/* columna Num de una matriz [C|R] */
getCol([],_,[]).
getCol([C|R],Num,[C1|R1]):-
getElem(Num,C,C1),
getCol(R,Num,R1)
.
/* fila Num de una matriz [C|R] */
getFil(L,Num,L1):-
getElem(Num,L,L1)
```

```
.

/* multiplicar matrices */
crearFila(_,Col,[],_,M2):-
cantCol(M2,Cant), Cant= Col,!
.

crearFila(Fil,Col,[C|R],M1,M2):-
getFil(M1,Fil,Fila),
getCol(M2,Col,Columna),
prodEscalar(Fila,Columna,C),
ColTemp is Col +1, crearFila(Fil,ColTemp,R,M1,M2)
.

mult(Fil,[],M1,_):-
cantFil(M1,Cant),Cant= Fil,
!.
mult(Fil,[C|R],M1,M2):-
crearFila(Fil,0,C,M1,M2),
FilTemp is Fil +1,
mult(FilTemp,R,M1,M2)
.

multiplicar(M1,M2,M):-
mult(0,M,M1,M2)
.
```

```
/* cantidad que se repite X en una lista*/

cantRep(_,[],0).


cantRep(X,[X|R],Cant):-

cantRep(X,R,Cant1),

Cant is Cant1+1,!.

cantRep(X,[_|R],Cant):-

cantRep(X,R,Cant)

.


/* jkkjhk no pincha */


mayr([X],1,X).


mayr([C|R],Cant,Elem):-

cantRep(C,[C|R],Cant1),

mayr(R,Cant2,Elem1),

(((Cant1>= Cant2) ,(Cant is Cant1,Elem is C)) ;( Cant is Cant2 , Elem is Elem1))

.

/* concatenar dos listas */


concat([],L,L).

concat([C|R],L,[C|R1]):-

concat(R,L,R1)

.
```

```
/* obtener todos los elementos atomicos de una lista de listas de listas de... */

flatten([],[]):-!.

flatten([C|R],L):-

flatten(C,L1),

flatten(R,L2),

concat(L1,L2,L),!

.

flatten(X,[X]).


/* suma de matrices */

crearfila([],[],[]).

crearfila([C|R],[C1|R1],[C2|R2]):-

C2 is C + C1,

crearfila(R,R1,R2)

.

sumaMat([],[],[]).

sumaMat([C|R],[C1|R1],[C2|R2]):-

crearfila(C,C1,C2),

sumaMat(R,R1,R2)

.

/* elemento de una matriz */

elemMat(Fila,Col,[C|R],X):-

getElem(Fila,[C|R],L),

getElem(Col,L,X)

.
```

```
/* sobreescribir un elemento en una lista */

sobreescribirEn(_,_,[],[]).

sobreescribirEn(Elem,0,[_|R],[Elem|R1]):-

sobreescribirEn(Elem,-1,R,R1),!

.

sobreescribirEn(Elem,Pos,[C|R],[C|R1]):-

ColTemp is Pos -1,

sobreescribirEn(Elem,ColTemp,R,R1)

.


/* sobreescribir un elemnto en una matriz */


sobreescribirMat(_,_,_,[],[]):-!.

sobreescribirMat(0,Col,Elem,[C|R],[C1|R1]):-

sobreescribirEn(Elem,Col,C,C1), FilTemp is -1,

sobreescribirMat(FilTemp,Col,Elem,R,R1),!

.

sobreescribirMat(Fil,Col,Elem,[C|R],[C|R1]):-

FilTemp is Fil - 1,

sobreescribirMat(FilTemp,Col,Elem,R,R1)

.


/* intercambiar elemntos de una matriz */

exchange(pto(Fila1,Col1),pto(Fila2,Col2),[C|R],[C1|R1]):-

elemMat(Fila1,Col1,[C|R],Pos1),

elemMat(Fila2,Col2,[C|R],Pos2),
```

```
sobreescribirMat(Fila1,Col1,Pos2,[C|R],M),
sobreescribirMat(Fila2,Col2,Pos1,M,[C1|R1])
.



/* sublistas de una lista*/
prim3([C,C1,C2|[]],[C,C1,C2|_]).

sublista3([],[]).
sublista3([C|R],[C1|R1]):-
prim3([C|R],[C1|R1]);
sublista3([C|R],R1)

% Ejercicio 4
% Definir los siguientes predicados:

% i. last(-L, -U), donde U es el ´ultimo elemento de la lista L. Definirlo en forma recursiva, y usando
% el predicado append definido de la siguiente manera:
% append([], X, X).
% append( [H | T], Y, [H | Z]) :- append(T, Y, Z).

last(L, U) :- append(_, [U], L).

% ii. reverse(+L, -L1), donde L1 contiene los mismos elementos que L, pero en orden inverso.
% Ejemplo: reverse([a,b,c], [c,b,a]).
```

```
% Realizar una definici´on usando append, y otra sin usarlo. Mostrar el ´arbol de prueba para el
% ejemplo dado.

% reverse([], []).
% reverse([H | T], [ReversedT | [H]]) :- reverse(T, ReversedT).
% reverse: como se hace reverse sin append?

reverse2([], []).
reverse2([H | T], R) :- reverse2(T, ReversedT), append(ReversedT, [H], R).

% iii. maxlista(+L, -M) y minlista(+L, -M), donde M es el m´aximo/m´inimo de la lista L.
maxlista([H], H).
maxlista([H | T], H) :- maxlista(T, M), H >= M.
maxlista([H | T], M) :- maxlista(T, M), H < M.

minlista([H], H).
minlista([H | T], H) :- minlista(T, M), H =< M.
minlista([H | T], M) :- minlista(T, M), H > M.

% iv. palindromo(+L, -L1), donde L1 es un pal´indromo construido a partir de L.
% Ejemplo: palindromo([a,b,c], [a,b,c,c,b,a]).
palindromo(L, P) :- reverse2(L, RevL), append(L, RevL, P).

% v. doble(-L, -L1), donde cada elemento de L aparece dos veces en L1.
% Ejemplo: doble([a,b,c], [a,a,b,b,c,c]).
doble([], []).
```

```
doble([H | T], [H | H | DT]) :- doble(T, DT).


% vi. prefijo(-P, +L), donde P es prefijo de la lista L.

prefijo(P, L) :- append(P, _, L).


% vii. sufijo(-S, +L), donde S es sufijo de la lista L.

sufijo(P, L) :- append(_, P, L).


% viii. sublista(-S, +L), donde S es sublista de L.

% Esta version es para sublistas "continuas" en L.

sublista([], _).

sublista(S, L) :- append(CasiL, _, L), append(_, S, CasiL), S \= [].


% Esta version es para sublistas "no continuas" en L.


sublistaB([], []).

sublistaB([H | SubsT], [H | T]) :- sublistaB(SubsT, T).

sublistaB(SubsT, [_ | T]) :- sublistaB(SubsT, T).


% ix. iesimo(-I, +L, -X), donde X es el I-´esimo elemento de la lista L.

% Ejemplo: iesimo(2, [10, 20, 30, 40], 20).

iesimo(I, L, X) :- append(ListI, [X | _], L), length(ListI, I).


% Definir los siguientes predicados:

% i. mezcla(L1, L2, L3), donde L3 es el resultado de mezclar uno a uno los elementos de las listas

% L1 y L2. Si una lista tiene longitud menor, entonces el resto de la lista m´as larga es pasado
```

```
% sin cambiar. Verificar la reversibilidad, es decir si es posible obtener L3 a partir de L1 y L2, y
% viceversa.
% Ejemplo: mezcla([a,b,c], [d,e], [a,d,b,e,c]).
mezcla(L1, [], L1).
mezcla([], L2, L2).
mezcla([H1 | T1], [H2 | T2], [H1, H2 | T12]) :- mezcla(T1, T2, T12).

% ii. split(N, L, L1, L2), donde L1 tiene los N primeros elementos de L, y L2 el resto. Si L tiene
% menos de N elementos el predicado debe fallar. ¿Cu´an reversible es este predicado? Es decir,
% ¿qu´e elementos pueden estar indefinidos al momento de la invocaci´on?
split(N, L, L1, L2) :- append(L1, L2, L), length(L1, N).

% iii. borrar(+ListaConXs, +X, -ListaSinXs), que elimina todas las ocurrencias de X de la lista
% ListaConXs.
borrar([], _, []).
borrar([H | XS], E, [H | YS]) :- E \= H, borrar(XS, E, YS).
borrar([E | XS], E, YS) :- borrar(XS, E, YS).

% iv. sacarDuplicados(+L1, -L2), que saca todos los elementos duplicados de la lista L1.
pertenece(E, L1) :- append(Principio, _, L1), append(_, [E], Principio).
sacarDuplicados([], []).
sacarDuplicados([H | T], [H | SinDups]) :- borrar(T, H, BorrarH), sacarDuplicados(BorrarH, SinDups).
sacarDuplicados([H | T], SinDups) :- pertenece(H, T), sacarDuplicados(T, SinDups).

%Ejecicio 7
%Definir el predicado aplanar(+Xs, -Ys), que es verdadero sii Ys contiene los elementos contenidos
```

%en alg´un nivel de Xs, en el mismo orden de aparici´on. Los elementos de Xs son enteros, ´atomos o

%nuevamente listas, de modo que Xs puede tener una profundidad arbitraria. Por el contrario, Ys es

%una lista de un solo nivel de profundidad.

%Ejemplos:

%?- aplanar([a, [3, b, []], [2]], [a, 3, b, 2]).

%?- aplanar([[1, [2, 3], [a]], [[[]]]], [1, 2, 3, a]).

aplanar([], []).

aplanar([H | T], App) :- is_list(H), aplanar(H, AppH), aplanar(T, AppT), append(AppH, AppT, App).

aplanar([H | T], App) :- not(is_list(H)), aplanar(T, AppT), append([H], AppT, App).

%Ejecicio 8

%Definir los siguientes predicados:

%i. ordenada(+L), que ser´a cierta si los elementos de L est´an ordenados en forma ascendente.

ordenada([]).

ordenada([_]).

ordenada([X, Y | XS]) :- X =< Y, ordenada([Y | XS]).

%ii. quicksort(+L, -L1), donde L1 es el resultado de ordenar L por el m´etodo de quicksort, que

%consiste en dividir a L en 2 sublistas con los menores y mayores al primer elemento, ordenar cada

%una de ellas y luego proceder a concatenarlas.

quicksort_partir([], _, [], []).

quicksort_partir([H | T], E, [H | Menores], Mayores) :- (H =< E), quicksort_partir(T, E, Menores, Mayores).

```
quicksort_partir([H | T], E, Menores, [H | Mayores]) :- H > E, quicksort_partir(T, E, Menores, Mayores).

quicksort([], []).

quicksort([H | T], S) :- quicksort_partir(T, H, Menores, Mayores), quicksort(Menores, MenOrd),

quicksort(Mayores, MayOrd), append(MenOrd, [H | MayOrd], S).

%iii. inssort(+L, -L1), donde L1 es el resultado de ordenar L por el m´etodo de inserci´on, que

%consiste en insertar cada elemento en el lugar adecuado del resto de la lista ya ordenada.

insertar_ordenado([], E, [E]).

insertar_ordenado([H | T], E, [E, H | T]) :- E =< H.

insertar_ordenado([H | T], E, [H | S]) :- E > H, insertar_ordenado(T, E, S).

inssort([], []).

inssort([H | T], S) :- inssort(T, SortT), insertar_ordenado(SortT, H, S).

% Ejercicio 9

% Definir un predicado rotar(+L, +N, -R), tal que R sea la lista L rotada N posiciones (la rotaci´on se

% debe hacer hacia la derecha si N>0 y hacia la izquierda si N<0).

% Ejemplos:

% rotar([1, a, 2, b, 3], 3, X) debe dar como respuesta X = [2, b, 3, 1, a]

% rotar([1, a, 2, b, 3], -3, X) debe dar como respuesta X = [b, 3, 1, a, 2]

modulus(N, Modulus, Result) :- Result is ((N mod Modulus) + Modulus) mod Modulus.

rotar(L, N, R) :- length(L, Length), modulus(N, Length, Rotate), append(Init, Tail, L),
```

length(Tail, Rotate), append(Tail, Init, R).

% Ejercicio 10

% Definir un predicado que reciba una lista de n´umeros naturales y devuelva otra lista de n ´umeros

% naturales, en la que cada n´umero n de la primera lista aparezca repetido n veces en forma consecutiva,

% respetando su orden de aparici´on. Considerar que la lista original siempre est´a instanciada.

% Ejemplo: para la lista [2, 3, 1, 0, 2] la salida es [2, 2, 3, 3, 3, 1, 2, 2].

repetir_n(_, 0, []).

repetir_n(E, Times, [E | T]) :- Times > 0, TimesMenos1 is Times - 1, repetir_n(E, TimesMenos1, T).

repetir_numeros([], []).

repetir_numeros([H | T], Reps) :- repetir_n(H, H, Hs), repetir_numeros(T, Ts), append(Hs, Ts, Reps).

% Ejercicio 11

% Escribir en Prolog un predicado que devuelva la mediana de una lista (la mediana es el elemento que

% se halla en la posici´on del medio de dicha lista, tras ser ordenada). Utilizar los predicados definidos

% anteriormente. Considerar que la lista siempre est´a instanciada.

mediana(L, M) :- quicksort(L, S), length(L, Length), Mitad is Length // 2,

append(Primeros, [M | _], S), length(Primeros, Mitad).

% Ejercicio 12

% Escribir en Prolog los siguientes predicados:

% interseccion(+X, +Y, -Z), tal que Z es la intersecci´on sin repeticiones de las listas X e Y,

% % respetando en Z el orden en que aparecen los elementos en X.

interseccion_con_duplicados([], _, []).

interseccion_con_duplicados([H | T], Y, [H | IntTY]) :- pertenece(H, Y),
interseccion_con_duplicados(T, Y, IntTY).

interseccion_con_duplicados([H | T], Y, IntTY) :- not(pertenece(H, Y)),
interseccion_con_duplicados(T, Y, IntTY).

interseccion(X, Y, Z) :- interseccion_con_duplicados(X, Y, W), sacarDuplicados(W, Z).

% Quicksort

partition([], _, [], []).

partition([X|Xs], Pivot, Smalls, Bigs) :- (X @< Pivot -> Smalls = [X|Rest], partition(Xs, Pivot, Rest, Bigs); Bigs = [X|Rest], partition(Xs, Pivot, Smalls, Rest)).

quicksort([]) --> [].

quicksort([X|Xs]) --> { partition(Xs, X, Smaller, Bigger) }, quicksort(Smaller), [X], quicksort(Bigger).

---

hanoi(N):- move(N,'A','C','B').

move(0,_,_,_):-!.

move(N, Src, Dest, Spare):- M is N-1, move(M, Src, Spare, Dest), primitive(Src, Dest), move(M, Spare, Dest, Src).

```
primitive(X, Y):- writelist([mueve, un, disco, desde, X, hasta, Y]), nl.

writelist([]).

writelist([H|T]):- write(H), write(' '), writelist(T).

% ********************************** OTRA VERSION
**********************************

hanoi(N) :- move(N, left, right, center).

move(0, _, _, _) :- !.

move(N, A, B, C) :- M is N-1, move(M, A, C, B), notify([A,B]), move(M, C, B, A).

% ********************************** OTRA VERSION
**********************************

/*

  The Towers of Hanoi - Copyright (c) Brian D Steel - 11 Dec 98 / 10 Feb 99

  ============================================================================

  In this classic children's puzzle, a pile of discs is stored on the

  left three poles, with each disc being smaller than the one beneath it.

  The object of the game is to transfer the whole pile to the right hand

  pole, one disc at a time, with the help of the middle pole, but at no

  stage may a larger disc be placed on a smaller one.
```

The idea is simple. Suppose you have a pile of discs on the LEFT pole,
and want to transfer them legally to the RIGHT pole. If you can somehow
transfer all but the very bottom disc to the MIDDLE pole, then all you
you need to do is move the remaining disc from LEFT to RIGHT, and then
somehow transfer the other discs from MIDDLE to RIGHT. The same is true
of any pile of discs on any one pole that you want on another pole:
transfer all but the last disc to a spare pole, move the last one
directly to your chosen target, and then transfer everything from the
spare pole to your target.

This is a classic candidate for recursion: figure out how to do one step,
as described above, and then trust a recursive call to handle the rest.

For example, to transfer 3 discs from LEFT to RIGHT, via MIDDLE, type:

```
?- hanoi(3).
Move disc from LEFT to RIGHT pole
Move disc from LEFT to MIDDLE pole
Move disc from RIGHT to MIDDLE pole
Move disc from LEFT to RIGHT pole
Move disc from MIDDLE to LEFT pole
Move disc from MIDDLE to RIGHT pole
Move disc from LEFT to RIGHT pole
yes
*/
```

```
% solve the towers of hanoi puzzle for the given number of discs

hanoi( Number ) :- hanoi( Number, 'LEFT', 'RIGHT', 'MIDDLE' ).

% if there is just one disc, simply move it across

hanoi( 1, Source, Target, _ ) :- move( Source, Target ), !.

% otherwise solve for one less disc than we have, using the spare pole

hanoi( Number, Source, Target, Spare ) :- Less is Number - 1, hanoi( Less, Source, Spare, Target ),
move( Source, Target ), hanoi( Less, Spare, Target, Source ).

% move a single disc from one pole to another

move( Source, Target ) :- write( 'Move disc from ' ), write( Source ), write( ' to ' ), write( Target ),
write( ' pole' ), nl.
```

MIGUEL SCHPEIR – UNIVERSIDAD NACIONAL DEL LITORAL