

Escuela Técnica Superior de Ingeniería Informática

Selección de Ejercicios de Programación Lógica

CURSO 2024-2025

**Autores: Joaquín Arias**

Copyright (c) 2024 Joaquín Arias. Esta obra está bajo la licencia CC BY-SA 4.0, [Creative Commons Atribuciónn-CompartirIgual 4.0 Inter-](https://creativecommons.org/licenses/by-sa/4.0/deed.es) [nacional](https://creativecommons.org/licenses/by-sa/4.0/deed.es).


# Índice

|  |  |  |
| --- | --- | --- |
| **1.** | **Aritmética de Peano** | [**3**](#_bookmark0) |
| **2.** | **Listas** | [**3**](#_bookmark4) |
| **3.** | **Árboles binarios** | [**7**](#_bookmark35) |
| **4.** | **Árbol genealógico** | [**7**](#_bookmark38) |
| **5.** | **Expresiones simbólicas** | [**9**](#_bookmark42) |
| **6.** | **Recursión de cola con acumulación** | [**10**](#_bookmark46) |
| **A.** | **Soluciones** | [**12**](#_bookmark48) |

**Agradecimientos**

Para elaborar esta obra nos hemos inspirado en transparencias, apuntes y compen- dios de ejercicios de diversas fuentes, incluyendo trabajos de: Ana Pradera (URJC’23), ClipLab (UPM’24), José A. Alonso Jiménez et al. (US’13).

# Aritmética de Peano

**Ejercicio 1.1** Implementa leq/2 a partir del predicado plus/3. Pista: un natural X es menor o igual que otro Y si existe un tercer natural Z tal que X+Z=Y.

(Sol. en pág. [12](#_bookmark49))

**Ejercicio 1.2** Implementa even(X), de modo que se cumple si X es un número natural par. Pista: implementarlo a partir del predicado plus/3.

(Sol. en pág. [12](#_bookmark50))

**Ejercicio 1.3** Implementa factorial(X,Y), de modo que se cumple si Y es el factorial de X: (i) usando la aritmética de Peano y (ii) usando los operadores aritméticos del sistema (p.ej., is/2).

(Sol. en pág. [12](#_bookmark51))

# Listas

Los ejercicios de esta sección corresponden al apartado dedicado a listas en el Ninety-Nine Prolog Problems escrito por Werner Hett de la Berne University of Ap- plied Sciences (Suiza).

**Ejercicio 2.1** my\_last(X,List) devuelve en X el último elemento de la lista List.

(Sol. en pág. [12](#_bookmark52))

**Ejercicio 2.2** lst\_but\_one(X,List) devuelve en \code{X en penúltimo ele- mento de la lista List.

(Sol. en pág. [12](#_bookmark53))

**Ejercicio 2.3** element(X,List,K) devuelve en K el K-ésimo elemento de la lista

List.

(Sol. en pág. [12](#_bookmark54))

**Ejercicio 2.4** my\_length(X, List) devuelve en X el número de elementos de la lista List.

(Sol. en pág. [12](#_bookmark55))

**Ejercicio 2.5** my\_reverse(List, Tsil) se cumple si Tsil es la lista List con los elementos ordenados de manera inversa.

(Sol. en pág. [13](#_bookmark56))

**Ejercicio 2.6** palindrome(List) se cumple si la lista List es un palíndromo (p.ej., “Anita lava la tina”). En listas se dice de aquella que tiene los mismos elementos hacia adelante que hacia atrás. (Pista: implementa usando reverse).

(Sol. en pág. [13](#_bookmark57))

**Ejercicio 2.7** my\_flatten(List,Flatten) se cumple si List es una lista de listas y Flatten es una lista con todos los elementos de las sublistas de List.

(Sol. en pág. [13](#_bookmark58))

**Ejercicio 2.8** compress(Liist,List) a partir de la lista Liist devuelve en List

el resultado de eliminar elementos duplicados consecutivos.

(Sol. en pág. [13](#_bookmark59))

**Ejercicio 2.9** pack(Liist,Pack) a partir de la lista Liist devuelve en Pack los elementos duplicados consecutivos dentro de sublistas.

(Sol. en pág. [13](#_bookmark60))

**Ejercicio 2.10** encode{Liist,Encode) aplicar el método de compresión de da- tos denominado codificación de longitud de ejecución: los duplicados consecutivos de elementos se codifican como términos [N,E] donde N es el número de duplicados del elemento E. (Pista, utiliza el resultado del problema de empaquetar duplicados conse- cutivos de elementos de lista en sublistas).

(Sol. en pág. [13](#_bookmark61))

**Ejercicio 2.11** encode\_modified (Liist, Encode\_B) modifica el resultado de co- dificación de longitud de ejecución de forma que si un elemento no tiene duplicados simplemente se copia en la lista resultante. Sólo los elementos con duplicados se trans- fieren como términos [N,E]. (Pista, utiliza el predicado encode/2 como predicado auxiliar).

(Sol. en pág. [14](#_bookmark62))

**Ejercicio 2.12** decode{Encode,Liist) a partir de Encode, una lista de códigos de longitud de ejecución generada como se especifica en el problema de codificación de longitud de ejecución (modificada o no), construya su versión descomprimida en Liist. 1.12 Decode a run-length encoded list.

(Sol. en pág. [14](#_bookmark63))

**Ejercicio 2.13** encode\_direct (Liist, Encode) implementa encode(Liist, Encode) de manera directa.

(Sol. en pág. [14](#_bookmark64))

**Ejercicio 2.14** duplicate(List, LLiisstt duplica los elementos de List.

(Sol. en pág. [14](#_bookmark65))

**Ejercicio 2.15** duplicate(List, N, N\_List) duplica los elementos de List, N

veces.

(Sol. en pág. [14](#_bookmark66))

**Ejercicio 2.16** drop(List,N,Lst) elimina el N-ésimo elemento de la lista List.

(Sol. en pág. [14](#_bookmark67))

**Ejercicio 2.17** split(List,N,Li,St) divide la lista List en dos partes, Li y St

de modo que la longitud de Li es N.

(Sol. en pág. [15](#_bookmark68))

**Ejercicio 2.18** slice(List, From, To, Is) extrae de List una sublista Is des- de el elemento en la posición From hasta el elemento en la posición To.

(Sol. en pág. [15](#_bookmark69))

**Ejercicio 2.19** rotate(List, N, Istl) desplaza los elementos de List hacia la izquierda N posiciones.

(Sol. en pág. [15](#_bookmark70))

**Ejercicio 2.20** remove\_at(X, List, K, Lst) elimina el elemento K-ésimo de

List y devuelve dicho elemento en X, y la lista resultante en Lst.

(Sol. en pág. [15](#_bookmark71))

**Ejercicio 2.21** insert\_at(A, List, K, Liast) inserta un elemento A en List

en la posición K-ésima y devuelve la lista resultante en Lst.

(Sol. en pág. [15](#_bookmark72))

**Ejercicio 2.22** range(From,To,List) crea una lista List con todos los enteros entre From y To.

(Sol. en pág. [16](#_bookmark73))

**Ejercicio 2.23** rnd\_select(List, N, Rs) extrae de List y devuelve en una lista Rs un total de N elementos seleccionados de manera random (usa el operador aritmético random(X), que es un (pseudo-)random entero en el rango [0,X]).[1](#_bookmark28)

(Sol. en pág. [16](#_bookmark74))

1Este operador esta disponible en Swi pero no en Ciao. En Ciao hay que usar el predicado random/1, que genera números (pseudo-)random en el intervalo [0.0,1.0].

**Ejercicio 2.24** lotto(N,M,Lotto) extrae N números aleatorios diferentes del con- junto desde 1 a M y los devuelve en Lotto. (Pista: usa como predicados auxiliares range/3 y rnd\_select/3).

(Sol. en pág. [16](#_bookmark75))

**Ejercicio 2.25** rnd\_permutation (List, Sitl) genera Sitl, una permutación aleatoria de elementos de List.

(Sol. en pág. [16](#_bookmark76))

**Ejercicio 2.26** combination(K, List, SubList) Generar las combinaciones de K elementos distintos elegidos entre los N elementos de la lista List. (Pista, imple- mentar con y sin predicado auxiliar)

(Sol. en pág. [16](#_bookmark77))

**Ejercicio 2.27** group2(List,G1,G2,G3) agrupa los elementos de un conjunto en subconjuntos disjuntos. a) ¿De cuántas formas puede funcionar un grupo de 9 personas en 3 subgrupos disjuntos de 2, 3 y 4 personas?

(Sol. en pág. [17](#_bookmark78))

**Ejercicio 2.28** group(List,Ns,Groups) generaliza el predicado group3/4 de for- ma que se puede especificar una lista de tamaños de grupos Ns y el predicado devuel- va una lista de grupos Groups. Ten en cuenta que no queremos permutaciones de los miembros del grupo; es decir, [[aldo,beat],...] es la misma solución que [[beat,aldo],...]. Sin embargo, hacemos una diferencia entre [[aldo,beat],[carla,david],...] y [[car- la,david],[aldo,beat],...].

Puedes encontrar más información sobre este problema combinatorio en un buen libro de matemáticas discretas bajo el término “coeficientes multinomiales”. (Sugeren- cia, utiliza el predicado substract/3).

(Sol. en pág. [17](#_bookmark79))

**Ejercicio 2.29** lsort(InList,Sort) Ordenar una lista de listas según la longitud de las sublistas:

1. Supongamos que una lista InList contiene elementos que a su vez son listas. El objetivo es ordenar los elementos de InList en función de su longitud. Por ejemplo, las listas cortas primero, las largas después, o vice- versa. (Pista, utiliza los predicados predefinidos keysort/2 y length/2). Ejempo:

?- lsort([[a,b,c],[d,e],[f,g,h],[d,e],[i,j,k,l],[m,n],[o]],L).

L = [[o],[d,e],[d,e],[m,n],[a,b,c],[f,g,h],[i,j,k,l]]

1. lfsort(InList,Sort) ahora suponemos que una lista InList contiene elementos que a su vez son listas. Pero esta vez el objetivo es ordenar los

elementos de InList según su frecuencia de longitud; es decir, en el caso por defecto, en el que la ordenación se hace de forma ascendente, las listas con longitudes poco frecuentes se colocan en primer lugar, otras con una longitud más frecuente vienen después. Ejemplo:

?- lfsort([[a,b,c],[d,e],[f,g,h],[d,e],[i,j,k,l],[m,n],[o]],L).

L = [[i,j,k,l],[o],[a,b,c],[f,g,h],[d,e],[d,e],[m,n]] ?

(Sol. en pág. [17](#_bookmark80))

# Árboles binarios

**Ejercicio 3.1** cbal\_tree(N, Tree) se cumple si el árbol binario Tree es comple- tamente equilibrados para un número N de nodos. El predicado debe generar todas las soluciones mediante backtracking. Ponga la letra ’x’ como información en todos los nodos del árbol.

En un árbol binario completamente equilibrado, se cumple la siguiente propiedad para cada nodo: El número de nodos de su subárbol izquierdo y el número de nodos de su subárbol derecho son casi iguales (su diferencia no es mayor que uno). Ejemplo:

?- cbal\_árbol(4,T).

T = tree(x,tree(x,void,void),tree(x,void,tree(x,void,void))) ?; T = tree(x,tree(x,void,void),tree(x,tree(x,void,void),void)) ?; etc no

(Sol. en pág. [18](#_bookmark81))

**Ejercicio 3.2** symmetric(Tree) se cumple si un árbol binario dado es simétrico. Un árbol binario es simétrico si se puede trazar una línea vertical a través del nodo raíz y entonces el subárbol derecho es la imagen especular del subárbol izquierdo. (Pis- ta, escribe primero un predicado mirror/2 para comprobar si un árbol es la imagen especular de otro. Sólo nos interesa la estructura, no el contenido de los nodos).

(Sol. en pág. [19](#_bookmark82))

# Árbol genealógico

**Ejercicio 4.1** Dado el siguiente programa:

1 % progenitor(?X, ?Y): se cumple si X es progenitor (madre o padre de Y

2 progenitor(pepa, pepe1).

3 progenitor(pepe, pepe1).

4 progenitor(pepe, pepa2).

5 progenitor(pepe1, pepa11).

6 progenitor(pepe1, pepa12).

7 progenitor(pepa12, pepe121).

8 % mujer(X): se cumple si X es una mujer

9 mujer(pepa).

10 mujer(pepa2).

11 mujer(pepa11).

12 mujer(pepa12).

13 % varon(X): se cumple si X es un varón

14 varon(pepe).

15 varon(pepe1).

16 varon(pepe121).

realice las siguiente consultas usando únicamente progenitor/2, mujer/1 y

varon/1:

1. ¿Es pepa la madre de pepa2?
2. ¿Quiénes son los progenitores de pepe1?
3. ¿Quién es la madre de pepe1?
4. ¿Tiene pepe1 hijos varones? ¿Quién o quiénes son?
5. ¿Quiénes son mujeres?
6. ¿Tiene pepa alguna nieta? ¿Quién o quiénes son?[2](#_bookmark41)
7. ¿Quiénes son progenitores comunes de pepa11 y pepa12?
8. ¿Tienen pepa11 y pepa12 algún progenitor/a común (no importa quién o quiénes)?

(Sol. en pág. [19](#_bookmark83))

**Ejercicio 4.2** A partir de los predicados progenitor/2, mujer/1 y varon/1, define los siguientes predicados (Pista: puedes usar predicados anteriores para implementar los siguientes):

1. madre(X,Y): se cumple si X es madre de Y.
2. madre(X): se cumple si X es madre.
3. hija(X,Y): se cumple si X es hija de Y.
4. abuelo(X,Y): se cumple si X es abuelo (varón) de Y.
5. hermana(X,Y): se cumple si X es hermana de Y. Pista: Al implementar es- te predicado considere que para que dos personas sean hermanas basta con

2OJO: Prolog debe mostrar solo las nietas de pepa, sus progenitores no deben aparecer. Recuerde el uso de variables semianónimas para resolver este tipo de situaciones.

que tengan un progenitor en común. Tenga en cuenta además que nadie es hermana de sí misma, es decir, la consulta ?- hermana(pepa,pepa) de- be devolver falso. El comportamiento anterior se puede conseguir usando adecuadamente el predicado predefinido X \= Y (que se cumple si X y Y no unifican).

1. tia(X,Y): se cumple si X es tía de Y.
2. ancestro(X,Y): se cumple si X es un ancestro de Y.
3. pariente(X,Y): se cumple si X es pariente de Y, es decir:

X es ancestro de Y (X es padre/abuelo/etc, de Y), o bien

 Y es ancestro de X (X es hijo/nieto/etc de Y), o bien

X e Y tienen un ancestro común (X e Y son hermanos, o primos, etc).

(Sol. en pág. [20](#_bookmark84))

# Expresiones simbólicas

**Ejercicio 5.1** Tablas de verdad para expresiones lógicas.

Definir predicados and/2, or/2, nand/2, nor/2, xor/2, impl/2 y equ/2 (para equivalencia lógica) que tienen éxito o fracasan según el resultado de sus respectivas operaciones; por ejemplo, and(A,B) tendrá éxito, si y sólo si tanto A como B tienen éxito. Nótese que A y B pueden ser objetivos (no sólo las constantes true y fail). Una expresión lógica en dos variables puede entonces escribirse en notación prefija, como en el siguiente ejemplo: and(or(A,B),nand(A,B)).

A continuación implementar el predicado table(A,B,Expr) que imprime la tabla de verdad de Exp en dos variables A y B. Ejemplo:

?- table(A,B,and(A,or(A,B))). true true true

true fail true fail true fail fail fail fail

(Sol. en pág. [20](#_bookmark85))

**Ejercicio 5.2** Tablas de verdad para expresiones lógicas (2). Extienda el progra- ma table/3 definiendo and/2, or/2, etc. como operadores. (Pista, usa la directiva

:- op( Preference, Position,Name)). Esto permite escribir la expresión lógica de

forma más natural, como en el ejemplo: A and (A or not B). Defina la preceden- cia de los operadores como de costumbre, es decir, como en Java. Ejemplo:

?- table(A,B, A and (A or not B)). true true true

true fail true fail true fail fail fail fail

(Sol. en pág. [20](#_bookmark86))

**Ejercicio 5.3** Tablas de verdad para expresiones lógicas (3). Generaliza el programa table/3 para que la expresión lógica puede contener cualquier número de variables lógicas.

Defina table/2 de forma que table(List,Expr) imprima la tabla de verdad de la expresión Expr, que contiene las variables lógicas enumeradas en List. Ejemplo

?- table([A,B,C], A and (B or C) equ A and B or A and C). true true true true

true true fail true true fail true true true fail fail true fail true true true fail true fail true fail fail true true fail fail fail true

(Sol. en pág. [21](#_bookmark87))

# Recursión de cola con acumulación

**Ejercicio 6.1** Implementa fibonacci(N,F), que se cumple si F es el N-ésimo nú- mero de Fibonacci, siendo *f*0 = 0, *f*1 = 1 y *fn* = *fn−*1 + *fn−*2. Crea una versión con recursión doble y otra con recursión por cola.

(Sol. en pág. [21](#_bookmark88))

**Ejercicio 6.2** Re-implementa los siguientes programas con recursión por cola:

* 1. exp(M, N, E): se cumple si E es igual a M elevado a N:

1 exp(\_, 0, 1).

2 exp(M, N, E) :-

3 N>0, N1 is N-1,

4 exp(M, N1, E1),

5 E is M\*E1.

* 1. num\_t(N,T): se cumple si T es el número triangular asociado con N, i.e., es la suma de todos los números naturales menores o iguales a N:

1 num\_t(1,1).

2 num\_t(N, T) :-

3 N>1, N1 is N-1,

4 num\_t(N1, T1),

5 T is T1+N.

# Soluciones

## Solución del ejercicio [1.1](#_bookmark1)

1 leq(X,Y) :- plus(X,\_,Y).

## Solución del ejercicio [1.2](#_bookmark2)

1 even(X) :- plus(Y,Y,X).

## Solución del ejercicio [1.3](#_bookmark3)

 Versión con notación de Peano:

1 % factorial(N, F): F equals N factorial

2 factorial(0, s(0)).

3 factorial(s(N), F) :-

4 factorial(N,F1),

5 times(s(N),F1,F).

 Versión con is/2 y otros operadores aritméticos del sistema:

1 % factorial(X, Y): se cumple si Y es el factorial

2 % del número natural X.

3 factorial(0, 1).

4 factorial(X, Y) :-

5 X > 0,

6 Z is X - 1,

7 factorial(Z, FZ),

8 Y is X\*FZ.

## Solución del ejercicio [2.1](#_bookmark5)

1 my\_last(X,[X]) :- !.

2 my\_last(X,[\_|T]) :- my\_last(X,T).

## Solución del ejercicio [2.2](#_bookmark6)

1 lst\_but\_one(X, [X,\_]) :- !.

2 lst\_but\_one(X, [\_|T]) :- lst\_but\_one(X,T).

## Solución del ejercicio [2.3](#_bookmark7)

1 element\_at(X,[X|\_], 1) :- !.

2 element\_at(X,[\_|T], N) :-

3 M is N - 1,

4 element\_at(X, T, M).

## Solución del ejercicio [2.4](#_bookmark8)

1 my\_length(1, [\_]) :- !.

2 my\_length(X, [\_|T]) :-

3 my\_length(Y,T),

4 X is Y+1.

## Solución del ejercicio [2.5](#_bookmark9)

1 my\_reverse(L, R) :- my\_reverse(L,[],R).

2 my\_reverse([H],L,[H|L]) :- !.

3 my\_reverse([X|T],L,R) :- my\_reverse(T,[X|L],R).

## Solución del ejercicio [2.6](#_bookmark10)

1 palindrome(L) :- reverse(L,L).

## Solución del ejercicio [2.7](#_bookmark11)

1 my\_flatten([H], X) :-

2 is\_list(H),

3 my\_flatten(H,X).

4 my\_flatten([H], [H]).

5 my\_flatten([H|T], X) :-

6 is\_list(H),

7 my\_flatten(H, Y),

8 my\_flatten(T, Z),

9 append(Y,Z,X).

10 my\_flatten([H|T], [H|X]) :- my\_flatten(T,X).

## Solución del ejercicio [2.8](#_bookmark12)

1 compress([H], [H]).

2 compress([H,H|T], X) :- compress([H|T],X).

3 compress([H|T], [H|X]) :- compress(T, X).

## Solución del ejercicio [2.9](#_bookmark13)

1 pack([], []).

2 pack([H,H|T], X) :- pack([[H,H]|T],X).

3 pack([[H|Hs]|[H|T]], X) :- pack([[H,H|Hs]|T], X).

4 pack([H|T], [[H]|X]) :- \+ is\_list(H), pack(T,X).

5 pack([H|T], [H|X]) :- pack(T,X).

## Solución del ejercicio [2.10](#_bookmark14)

1 encode([], []) :- !.

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

3 pack(L,[[X|Xs]|T]),

4 length([X|Xs],N), !,

5 encode(T, Y).

## Solución del ejercicio [2.11](#_bookmark15)

1 encode\_modified([], []).

2 encode\_modified(L,X) :- encode(L, Y), enc\_mod(Y,X).

3

4 enc\_mod([], []).

5 enc\_mod([[1,H]|T], [H|X]) :- enc\_mod(T, X), !.

6 enc\_mod([H|T], [H|X]) :- enc\_mod(T,X).

## Solución del ejercicio [2.12](#_bookmark16)

1 decode([], []).

2 decode([[1,H]|T], [H|X]) :- decode(T,X), !.

3 decode([[N,H]|T], [H|X]) :- M is N-1, decode([[M,H]|T], X), !.

4 decode([H|T], [H|X]) :- decode(T, X).

## Solución del ejercicio [2.13](#_bookmark17)

1 encode\_direct([], []).

2 encode\_direct([H,H|T], X) :-

3 encode\_direct([[2,H]|T], X), !.

4 encode\_direct([[N,H],H|T], X) :-

5 M is N + 1,

6 encode\_direct([[M,H]|T], X).

7 encode\_direct([H|T], [H|X]) :-

8 encode\_direct(T, X).

## Solución del ejercicio [2.14](#_bookmark18)

1 duplicate([H], [H,H]).

2 duplicate([H|L], [H,H|X]) :- duplicate(L, X).

## Solución del ejercicio [2.15](#_bookmark19)

1 duplicate([H], 1, [H]) :- !.

2 duplicate([H], N, [H|X]) :-

3 M is N - 1,

4 M > 0,

5 duplicate([H], M, X).

6 duplicate([H|T], N, X) :-

7 duplicate([H], N, Y),

8 duplicate(T, N, Z), !,

9 append(Y, Z, X).

## Solución del ejercicio [2.16](#_bookmark20)

1 drop(L, N, L) :-

2 length(L, Length),

3 Length < N.

4 drop(List, N, NthDropped) :-

5 length(List,Length),

6 Length >= N,

7 length(WithNth, N),

8 append(WithNth, Tail, List),

9 drop(Tail, N, Rest),

10 length(NthElem, 1),

11 append(WithoutNth, NthElem, WithNth),

12 append(WithoutNth, Rest, NthDropped).

## Solución del ejercicio [2.17](#_bookmark21)

1 split(L, 0, [], L).

2 split([H|T], N, [H|X], L2) :-

3 N > 0,

4 M is N - 1,

5 split(T, M, X, L2).

## Solución del ejercicio [2.18](#_bookmark22)

1 slice([H|\_], 1, 1, [H]).

2 slice([H|T], 1, To, [H|X]) :-

|  |  |  |
| --- | --- | --- |
| 3 | N is To - 1, |  |
| 4 | slice(T, 1, N, | X). |
| 5 | slice([\_|T], From, To, | L) :- |
| 6 | N is From - 1, |  |
| 7 | M is To - 1, |  |
| 8 | slice(T, N, M, | L). |

## Solución del ejercicio [2.19](#_bookmark23)

1 rotate([H|T], 1, X) :-

2 append(T, [H], X), !.

3 rotate([H|T],N,X) :-

4 append(T, [H], Y),

5 N1 is N - 1,

6 rotate(Y, N1, X).

## Solución del ejercicio [2.20](#_bookmark24)

1 ve\_at(H, [H|T], 1, T).

2 ve\_at(X, [H|T], N, [H|R]) :-

3 N1 is N - 1,

4 remove\_at(X, T, N1, R).

## Solución del ejercicio [2.21](#_bookmark25)

1 insert\_at(A, L, 1, [A|L]).

2 insert\_at(A, [H|T], N, [H|R]) :-

3 N1 is N - 1,

4 insert\_at(A, T, N1, R).

## Solución del ejercicio [2.22](#_bookmark26)

1 range(N, N, [N]) :- !.

2 range(N, M, [N|T]) :-

3 N1 is N + 1,

4 range(N1, M, T).

## Solución del ejercicio [2.23](#_bookmark27)

1 rnd\_select(L, 1, [H]) :-

2 length(L, Length),

3 R is random(Length) + 1,

4 remove\_at(H, L, R, \_).

5 rnd\_select(L, N, [H|T]) :-

6 length(L, Length),

7 R is random(Length) + 1,

8 remove\_at(H, L, R, X),

9 N1 is N - 1,

10 rnd\_select(X, N1, T).

## Solución del ejercicio [2.24](#_bookmark29)

1 lotto(N, M, X) :-

2 range(1, M, L),

3 rnd\_select(L, N, X).

## Solución del ejercicio [2.25](#_bookmark30)

1 rnd\_permutation(L, X) :-

2 length(L, S),

3 rnd\_select(L,S,X).

## Solución del ejercicio [2.26](#_bookmark31)

 Versión sin predicado auxiliar.

1 combination(1, [H|\_], [H]).

2 combination(N, [H|T], [H|C]) :-

3 N1 is N - 1, N1 > 0,

4 combination(N1, T, C).

5 combination(N, [\_|T], C) :- combination(N, T, C).

 Versión usando el predicado el/3. ¿Sabrías explicar que hace exactamente?

1 combination(0,\_,[]).

2 combination(K,L,[X|Xs]) :-

3 K > 0, el(X,L,R), K1 is K-1,

4 combination(K1,R,Xs).

5

6 el(X,[X|L],L).

7 el(X,[\_|L],R) :- el(X,L,R).

## Solución del ejercicio [2.27](#_bookmark32)

 Versión con los predicados auxiliares selectN/3 y substract/3 cuyo signi- ficado se puede inferir a partir del programa.

1 group3(G,G1,G2,G3) :-

2 selectN(2,G,G1),

3 subtract(G,G1,R1),

4 selectN(3,R1,G2),

5 subtract(R1,G2,R2),

6 selectN(4,R2,G3),

7 subtract(R2,G3,[]).

8

9 [...] % implementar selectN/3 y substract/3

 Versión usando el predicado forall/2 de [https://github.com/Rootex/](https://github.com/Rootex/99-prolog-problems/blob/master/lists.prolog) [99-prolog-problems/blob/master/lists.prolog](https://github.com/Rootex/99-prolog-problems/blob/master/lists.prolog)

1 group3(L, G1, G2, G3) :-

2 combination(2, L, G1),

3 combination(3, L, G2),

4 combination(4, L, G3),

5 forall(member(M1, G1),

6 forall(member(M2, G2),

7 forall(member(M3, G3),

8 (M1 \== M2,

9 M2 \== M3,

10 M1 \== M3)))).

## Solución del ejercicio [2.28](#_bookmark33)

1 group([],[],[]).

2 group(G,[N1|Ns],[G1|Gs]) :-

3 selectN(N1,G,G1),

4 subtract(G,G1,R),

5 group(R,Ns,Gs).

## Solución del ejercicio [2.29](#_bookmark34)

* 1. Implementación de lsort/2:

1 lsort(InList,OutList) :- lsort(InList,OutList,asc).

2

3 % sorting direction Dir is either asc or desc

4 lsort(InList,OutList,Dir) :-

5 add\_key(InList,KList,Dir),

6 keysort(KList,SKList),

7 rem\_key(SKList,OutList).

8

9 add\_key([],[],\_).

10 add\_key([X|Xs],[L-p(X)|Ys],asc) :- !,

11 length(X,L), add\_key(Xs,Ys,asc).

12 add\_key([X|Xs],[L-p(X)|Ys],desc) :-

13 length(X,L1), L is -L1, add\_key(Xs,Ys,desc).

14

15 rem\_key([],[]).

16 rem\_key([\_-p(X)|Xs],[X|Ys]) :- rem\_key(Xs,Ys).

* 1. Implementación de lfsort/2:

1 lfsort(InList,OutList) :- lfsort(InList,OutList,asc).

2

3 % sorting direction Dir is either asc or desc

4 lfsort(InList,OutList,Dir) :-

5 add\_key(InList,KList,desc),

6 keysort(KList,SKList),

7 pack(SKList,PKList),

8 lsort(PKList,SPKList,Dir),

9 flatten(SPKList,FKList),

10 rem\_key(FKList,OutList).

11

12 pack([],[]).

13 pack([L-X|Xs],[[L-X|Z]|Zs]) :- transf(L-X,Xs,Ys,Z), pack(Ys,Zs).

14

15 % transf(L-X,Xs,Ys,Z) Ys is the list that remains from the list Xs

16 % when all leading copies of length L are removed and transfed to Z

17 transf(\_,[],[],[]).

18 transf(L-\_,[K-Y|Ys],[K-Y|Ys],[]) :- L \= K.

19 transf(L-\_,[L-X|Xs],Ys,[L-X|Zs]) :- transf(L-X,Xs,Ys,Zs).

## Solución del ejercicio [3.1](#_bookmark36)

1 cbal\_tree(0,void) :- !.

2 cbal\_tree(N,tree(x,L,R)) :- N > 0,

3 N0 is N - 1,

4 N1 is N0//2, N2 is N0 - N1,

5 distrib(N1,N2,NL,NR),

6 cbal\_tree(NL,L), cbal\_tree(NR,R).

7

8 distrib(N,N,N,N) :- !.

9 distrib(N1,N2,N1,N2).

10 distrib(N1,N2,N2,N1).

## Solución del ejercicio [3.2](#_bookmark37)

1 symmetric(void).

2 symmetric(tree(\_,L,R)) :- mirror(L,R).

3

4 mirror(void,void).

5 mirror(tree(\_,L1,R1),tree(\_,L2,R2)) :- mirror(L1,R2), mirror(R1,L2).

## Solución del ejercicio [4.1](#_bookmark39)

1. ?- progenitor(pepa, pepa2), mujer(pepa).
2. ?- progenitor(X, pepe1).
3. ?- progenitor(X, pepe1), mujer(X).
4. ?- progenitor(pepe1, X), varon(X).
5. ?- mujer(X).
6. Las respuestas generadas por ?- progenitor(pepa,Y), progenitor(Y,N), mujer(N) son correctas (asocia a N las dos nietas de pepa, pepa11 y pepa12) pero facilita además los valores de los progenitores intermedios en la variable Y, información que no se requiere. Ejecute la siguiente consulta para comprobar que la variable anónima \_ NO sirve para solucionar este problema: La consulta debe mostrar solo las nietas de pepa, sus progenitores no deben aparecer:

La respuesta correcta es ?- progenitor(pepa,\_Y), progenitor(\_Y,N), mujer(N). Note que si en lugar de \_Y usamos la variable anónima \_ además de pepa11 y pepa12, facilita la respuesta pepa2, que no es nieta de Pepa. Esto se debe a que la variable anónima, a diferencia de \_Y, no tiene por qué unificar consigo misma, es decir, la primera y la segunda aparición de \_ pueden unificar con cosas distintas. Por ello, en la consulta anterior Prolog comprueba que Pepa es progenitora de alguien (no importa de quién) y luego comprueba que N es hija de alguien (tampoco importa de quién) y que es mujer, y estas condiciones también las cumple pepa2 (además de pepa11 y pepa12).

1. ?- progenitor(X, pepa11), progenitor(X, pepa12).
2. ?- progenitor(\_X, pepa11), progenitor(\_X, pepa12).

## Solución del ejercicio [4.2](#_bookmark40)

1. madre(X,Y) :- progenitor(X,Y), mujer(X).
2. madre(X) :- madre(X,\_).
3. hija(X,Y) :- progenitor(Y,X), mujer(X).
4. abuelo(X,Y) :- progenitor(X,Z), progenitor(Z,Y), varon(X).
5. hermana(X,Y) :-

progenitor(Z,X), progenitor(Z,Y), mujer(X), X\=Y.

1. tia(X,Y) :- progenitor(Z,Y), hermana(X,Z).
2. ancestro(X,Y) :- progenitor(X,Y).

ancestro(X,Y) :- progenitor(X,Z), ancestro(Z,Y).

1. pariente(X,Y) :- ancestro(X,Y).

pariente(X,Y) :- ancestro(Y,X).

pariente(X,Y) :- ancestro(Z, X), ancestro(Z, Y), X\=Y.

## Solución del ejercicio [5.1](#_bookmark43)

1 table(A,B,Expr) :- bind(A), bind(B), do(A,B,Expr), fail.

2

3 and(A,B) :- A, B.

4 or(A,\_) :- A.

5 or(\_,B) :- B.

6 equ(A,B) :- or(and(A,B), and(not(A),not(B))).

7 xor(A,B) :- not(equ(A,B)).

8 nor(A,B) :- not(or(A,B)).

9 nand(A,B) :- not(and(A,B)).

10 impl(A,B) :- or(not(A),B).

11

12 % bind(X) :- instantiate X to be true and false successively

13 bind(true).

14 bind(fail).

15

16 do(A,B,\_) :- display(A), display(' '),

17 display(B), display(' '), fail.

18 do(\_,\_,Expr) :- Expr, !, display(true), nl.

19 do(\_,\_,\_) :- display(fail), nl.

## Solución del ejercicio [5.2](#_bookmark44)

1 :- op(900, fy, not).

2 :- op(910, yfx, and).

3 :- op(910, yfx, nand).

4 :- op(920, yfx, or).

5 :- op(920, yfx, nor).

6 :- op(930, yfx, impl).

7 :- op(930, yfx, equ).

8 :- op(930, yfx, xor).

9

10 table(A,B,Expr) :- bind(A), bind(B), do(A,B,Expr), fail.

11 [...]

## Solución del ejercicio [5.3](#_bookmark45)

1 table(VarList,Expr) :- bindList(VarList), do(VarList,Expr), fail.

2

3 bindList([]).

4 bindList([V|Vs]) :- bind(V), bindList(Vs).

5

6 do(VarList,Expr) :- writeVarList(VarList), writeExpr(Expr), nl.

7

8 writeVarList([]).

9 writeVarList([V|Vs]) :- write(V), write(' '), writeVarList(Vs).

10

11 writeExpr(Expr) :- Expr, !, write(true).

12 writeExpr(\_) :- write(fail).

13 [...]

## Solución del ejercicio [6.1](#_bookmark47)

 Versión con doble recursión:

1 fibonacci(0,0).

2 fibonacci(1,1).

3 fibonacci(N,F) :-

4 N > 1,

5 N\_1 is N-1,

6 N\_2 is N-2,

7 fib(N\_1, FN\_1),

8 fib(N\_2, FN\_2),

9 F is FN\_1 + FN\_2.

 Versión con doble recursión por cola (con acumulador):

1 fibonacci(0,0).

2 fibonacci(1,1).

3 fibonacci(N,F) :- N > 1, fibonacci\_acc(1,1,1,N,F).

4

5 fibonacci\_acc(\_,F1,N,N,F1).

6 fibonacci\_acc(F0,F1,I,N,F) :-

7 N > I,

8 F\_2 is F0 + F1,

9 I2 is I + 1,

10 fibonacci\_acc(F1,F2,I2,N,F).