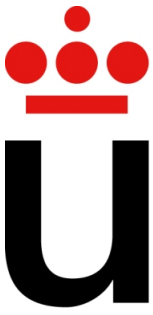


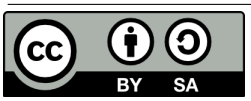
Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INFORMÁTICA

Selección de Ejercicios de Programación Lógica

CURSO 2024-2025

Autores: Joaquín Arias



Índice

Agradecimientos

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

1. 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)

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)

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)

2. 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 Applied Sciences (Suiza).

Ejercicio 2.1 `my_last(X,List)` devuelve en `X` el último elemento de la lista `List`.

(Sol. en pág. 12)

Ejercicio 2.2 `lst_but_one(X,List) devuelve en \code{X` en penúltimo elemento de la lista `List`.

(Sol. en pág. 12)

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

(Sol. en pág. 12)

Ejercicio 2.4 `my_length(X, List)` devuelve en `X` el número de elementos de la lista `List`.

(Sol. en pág. 12)

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)

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)

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)

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)

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)

Ejercicio 2.10 `encode{Liist,Encode)` aplicar el método de compresión de datos 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 consecutivos de elementos de lista en sublistas).

(Sol. en pág. 13)

Ejercicio 2.11 `encode_modified(Liist,Encode_B)` modifica el resultado de codificación 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 transfieren como términos [N,E]. (Pista, utiliza el predicado `encode/2` como predicado auxiliar).

(Sol. en pág. 14)

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)

Ejercicio 2.13 `encode_direct(Liist, Encode)` implementa `encode(Liist, Encode)` de manera directa.

(Sol. en pág. 14)

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

(Sol. en pág. 14)

Ejercicio 2.15 `duplicate(List, N, N_List)` duplica los elementos de `List`, `N` veces.

(Sol. en pág. 14)

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

(Sol. en pág. 14)

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)

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

(Sol. en pág. 15)

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

(Sol. en pág. 15)

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)

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)

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

(Sol. en pág. 16)

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]

(Sol. en pág. 16)

[1] Este 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 conjunto desde `1` a `M` y los devuelve en `Lotto`. (Pista: usa como predicados auxiliares `range/3` y `rnd_select/3`).

(Sol. en pág. 16)

Ejercicio 2.25 `rnd_permutation(List, Sitl)` genera `Sitl`, una permutación aleatoria de elementos de `List`.

(Sol. en pág. 16)

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

(Sol. en pág. 16)

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)

Ejercicio 2.28 `group(List,Ns,Groups)` generaliza el predicado `group3/4` de forma que se puede especificar una lista de tamaños de grupos `Ns` y el predicado devuelva 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 [[carla,david],[aldo,beat],...].

Puedes encontrar más información sobre este problema combinatorio en un buen libro de matemáticas discretas bajo el término "coeficientes multinomiales". (Sugerencia, utiliza el predicado `substract/3`).

(Sol. en pág. 17)

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

a) 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 viceversa. (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]]
```

b) `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)

3. Árboles binarios

Ejercicio 3.1 `cbal_tree(N, Tree)` se cumple si el árbol binario `Tree` es completamente 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)

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. (Pista, 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)

4. Árbol genealógico

Ejercicio 4.1 Dado el siguiente programa, que representa un esquema de parentesco simplificado de Abraham, basado en las relaciones familiares descritas en el Génesis:

```
1 % progenitor(?X, ?Y): se cumple si X es progenitor
2                        (madre, padre u otro) de Y
3 progenitor(terach,abraham).
4 progenitor(terach,nachor).
```

```
progenitor(terach,haran).
progenitor(abraham,isaac).
progenitor(haran,lot).
progenitor(haran,milcah).
progenitor(haran,yiscah).
progenitor(milcah,sarah).
progenitor(sarah,isaac).
% mujer(X): se cumple si X es una mujer
mujer(sarah).
mujer(milcah).
mujer(yiscah).
% varon(X): se cumple si X es un varón
varon(terach).
varon(abraham).
varon(nachor).
varon(haran).
varon(isaac).
varon(lot).
```

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

a) ¿Es sarah la madre de isaac?

b) ¿Quiénes son los progenitores de isaac?

c) ¿Quién es la madre de isaac?

d) ¿Tiene abraham hijos varones? ¿Quién o quiénes son?

e) ¿Quiénes son mujeres?

f) ¿Tiene terach alguna nieta? ¿Quién o quiénes son?.

Pista, use variables semianónimas para mostrar solo las nietas de terach (sus progenitores no deben aparecer).

g) ¿Quiénes son progenitores comunes de abraham y haran?

h) ¿Tienen abraham y haran algún progenitor/a común (no importa quién o quiénes)?

(Sol. en pág. 19)

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):

a) `madre(X,Y)`: se cumple si `X` es madre de `Y`.

b) `madre(X)`: se cumple si `X` es madre.

c) `hija(X,Y)`: se cumple si `X` es hija de `Y`.

d) `abuelo(X,Y)`: se cumple si `X` es abuelo (varón) de `Y`.

e) `hermana(X,Y)`: se cumple si `X` es hermana de `Y`. Pista: Al implementar este predicado considere que para que dos personas sean hermanas basta con que tengan un progenitor en común. Tenga en cuenta además que nadie es hermana de sí misma, es decir, la consulta `?- hermana(sarah,sarah)` debe devolver falso. El comportamiento anterior se puede conseguir usando adecuadamente el predicado predefinido `X \= Y` (que se cumple si `X` y `Y` no unifican).

f) `tia(X,Y)`: se cumple si `X` es tía de `Y`.

g) `ancestro(X,Y)`: se cumple si `X` es un ancestro de `Y`.

h) `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)

5. 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)

Ejercicio 5.2 Tablas de verdad para expresiones lógicas (2). Extienda el programa `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 precedencia 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)

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)

6. Recursión de cola con acumulación

Ejercicio 6.1 Implementa `fibonacci(N,F)`, que se cumple si `F` es el `N`-ésimo número de Fibonacci, siendo $f_0 = 0$, $f_1 = 1$ y $f_n = f_{n-1} + f_{n-2}$. Crea una versión con recursión doble y otra con recursión por cola.

(Sol. en pág. 21)

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

a) `exp(M, N, E)`: se cumple si `E` es igual a `M` elevado a `N`:

```
exp(_, 0, 1).
exp(M, N, E) :-
      N>0, N1 is N-1,
      exp(M, N1, E1),
      E is M*E1.
```

b) `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`:

```
num_t(1,1).
num_t(N, T) :-
      N>1, N1 is N-1,
      num_t(N1, T1),
      T is T1+N.
```

7. Misceláneas

Ejercicio 7.1 Implementa un programa para compilar usando el compilador de Ciao, `ciaoc` que permita calcular expresiones aritméticas usando `is/2`.

(Sol. en pág. 22)

Ejercicio 7.2 Implementa un programa para compilar usando el compilador de Ciao, `ciaoc` que permita calcular sistemas de ecuaciones lineales con el sistema de resoluciones sobre los racionales (clpq). Indica la invocación para resolver los siguientes sistemas de ecuaciones:

a) $\begin{cases} 3x+5y=2 \\ 5x+3y=-2 \end{cases}$

b) $\begin{cases} 2a+b-3c=7 \\ 5a-4b+c=-19 \\ a-b-4c=4 \end{cases}$

(Sol. en pág. 22)

A. Soluciones

Solución del ejercicio 1.1

```
leq(X,Y) :- plus(X,_,Y).
```

Solución del ejercicio 1.2

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

Solución del ejercicio 1.3

- Versión con notación de Peano:

```
% factorial(N, F): F equals N factorial
factorial(0, s(0)).
factorial(s(N), F) :-
      factorial(N,F1),
      times(s(N),F1,F).
```

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

```
% factorial(X, Y): se cumple si Y es el factorial
%                  del número natural X.
factorial(0, 1).
factorial(X, Y) :-
      X > 0,
      Z is X - 1,
      factorial(Z, FZ),
      Y is X*FZ.
```

Solución del ejercicio 2.1

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

Solución del ejercicio 2.2

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

Solución del ejercicio 2.3

```
element_at(X,[X|_], 1) :- !.
element_at(X,[_|T], N) :-
        M is N - 1,
        element_at(X, T, M).
```

Solución del ejercicio 2.4

```
my_length(1, [_]) :- !.
my_length(X, [_|T]) :-
        my_length(Y,T),
        X is Y+1.
```

Solución del ejercicio 2.5

```
my_reverse(L, R) :- my_reverse(L,[],R).
my_reverse([H],L,[H|L]) :- !.
my_reverse([X|T],L,R) :- my_reverse(T,[X|L],R).
```

Solución del ejercicio 2.6

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

Solución del ejercicio 2.7

```
my_flatten([H], X) :-
        is_list(H),
my_flatten(H,X).
my_flatten([H], [H]).
my_flatten([H|T], X) :-
        is_list(H),
        my_flatten(H, Y),
        my_flatten(T, Z),
        append(Y,Z,X).
my_flatten([H|T], [H|X]) :- my_flatten(T,X).
```

Solución del ejercicio 2.8

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

Solución del ejercicio 2.9

```
pack([], []).
pack([H,H|T], X) :- pack([[H,H]|T],X).
pack([[H|Hs]|[H|T]], X) :- pack([[H,H|Hs]|T], X).
pack([H|T], [[H]|X]) :- \+ is_list(H), pack(T,X).
pack([H|T], [H|X]) :- pack(T,X).
```

Solución del ejercicio 2.10

```
encode([], []) :- !.
encode(L, [[N,X]|Y]) :-
        pack(L,[[X|Xs]|T]),
        length([X|Xs],N), !,
        encode(T, Y).
```

Solución del ejercicio 2.11

```
encode_modified([], []).
encode_modified(L,X) :- encode(L, Y), enc_mod(Y,X).

enc_mod([], []).
enc_mod([[1,H]|T], [H|X]) :- enc_mod(T, X), !.
enc_mod([H|T], [H|X]) :- enc_mod(T,X).
```

Solución del ejercicio 2.12

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

Solución del ejercicio 2.13

```
encode_direct([], []).
encode_direct([H,H|T], X) :-
        encode_direct([[2,H]|T], X), !.
encode_direct([[N,H],H|T], X) :-
        M is N + 1,
        encode_direct([[M,H]|T], X).
encode_direct([H|T], [H|X]) :-
        encode_direct(T, X).
```

Solución del ejercicio 2.14

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

Solución del ejercicio 2.15

```
duplicate([H], 1, [H]) :- !.
duplicate([H], N, [H|X]) :-
        M is N - 1,
        M > 0,
        duplicate([H], M, X).
duplicate([H|T], N, X) :-
        duplicate([H], N, Y),
        duplicate(T, N, Z), !,
        append(Y, Z, X).
```

Solución del ejercicio 2.16

```
drop(L, N, L) :-
      length(L, Length),
      Length < N.
```

```
drop(List, N, NthDropped) :-
        length(List,Length),
        Length >= N,
        length(WithNth, N),
        append(WithNth, Tail, List),
        drop(Tail, N, Rest),
        length(NthElem, 1),
        append(WithoutNth, NthElem, WithNth),
        append(WithoutNth, Rest, NthDropped).
```

Solución del ejercicio 2.17

```
split(L, 0, [], L).
split([H|T], N, [H|X], L2) :-
          N > 0,
          M is N - 1,
          split(T, M, X, L2).
```

Solución del ejercicio 2.18

```
slice([H|_], 1, 1, [H]).
slice([H|T], 1, To, [H|X]) :-
          N is To - 1,
          slice(T, 1, N, X).
slice([_|T], From, To, L) :-
          N is From - 1,
          M is To - 1,
          slice(T, N, M, L).
```

Solución del ejercicio 2.19

```
rotate([H|T], 1, X) :-
          append(T, [H], X), !.
rotate([H|T],N,X) :-
          append(T, [H], Y),
          N1 is N - 1,
          rotate(Y, N1, X).
```

Solución del ejercicio 2.20

```
ve_at(H, [H|T], 1, T).
ve_at(X, [H|T], N, [H|R]) :-
          N1 is N - 1,
          remove_at(X, T, N1, R).
```

Solución del ejercicio 2.21

```
insert_at(A, L, 1, [A|L]).
```

```
insert_at(A, [H|T], N, [H|R]) :-
        N1 is N - 1,
        insert_at(A, T, N1, R).
```

Solución del ejercicio 2.22

```
range(N, N, [N]) :- !.
range(N, M, [N|T]) :-
        N1 is N + 1,
        range(N1, M, T).
```

Solución del ejercicio 2.23

```
rnd_select(L, 1, [H]) :-
        length(L, Length),
        R is random(Length) + 1,
        remove_at(H, L, R, _).
rnd_select(L, N, [H|T]) :-
       length(L, Length),
       R is random(Length) + 1,
       remove_at(H, L, R, X),
       N1 is N - 1,
       rnd_select(X, N1, T).
```

Solución del ejercicio 2.24

```
lotto(N, M, X) :-
        range(1, M, L),
        rnd_select(L, N, X).
```

Solución del ejercicio 2.25

```
rnd_permutation(L, X) :-
        length(L, S),
        rnd_select(L,S,X).
```

Solución del ejercicio 2.26

- Versión sin predicado auxiliar.

```
combination(1, [H|_], [H]).
combination(N, [H|T], [H|C]) :-
        N1 is N - 1, N1 > 0,
        combination(N1, T, C).
combination(N, [_|T], C) :- combination(N, T, C).
```

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

```
combination(0,_,[]).
combination(K,L,[X|Xs]) :-
```

```
    K > 0, el(X,L,R), K1 is K-1,
    combination(K1,R,Xs).

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

Solución del ejercicio 2.27

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

```
group3(G,G1,G2,G3) :-
    selectN(2,G,G1),
    subtract(G,G1,R1),
    selectN(3,R1,G2),
    subtract(R1,G2,R2),
    selectN(4,R2,G3),
    subtract(R2,G3,[]).

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

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

```
group3(L, G1, G2, G3) :-
        combination(2, L, G1),
        combination(3, L, G2),
        combination(4, L, G3),
        forall(member(M1, G1),
               forall(member(M2, G2),
                      forall(member(M3, G3),
                             (M1 \== M2,
                              M2 \== M3,
                              M1 \== M3)))).
```

Solución del ejercicio 2.28

```
group([],[],[]).
group(G,[N1|Ns],[G1|Gs]) :-
   selectN(N1,G,G1),
   subtract(G,G1,R),
   group(R,Ns,Gs).
```

Solución del ejercicio 2.29

a) Implementación de `lsort/2`:

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

% sorting direction Dir is either asc or desc
lsort(InList,OutList,Dir) :-
   add_key(InList,KList,Dir),
   keysort(KList,SKList),
   rem_key(SKList,OutList).

add_key([],[],_).
add_key([X|Xs],[L-p(X)|Ys],asc) :- !,
        length(X,L), add_key(Xs,Ys,asc).
add_key([X|Xs],[L-p(X)|Ys],desc) :-
        length(X,L1), L is -L1, add_key(Xs,Ys,desc).

rem_key([],[]).
rem_key([_-p(X)|Xs],[X|Ys]) :- rem_key(Xs,Ys).
```

b) Implementación de `lfsort/2`:

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

% sorting direction Dir is either asc or desc
lfsort(InList,OutList,Dir) :-
        add_key(InList,KList,desc),
   keysort(KList,SKList),
   pack(SKList,PKList),
   lsort(PKList,SPKList,Dir),
   flatten(SPKList,FKList),
   rem_key(FKList,OutList).

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

% transf(L-X,Xs,Ys,Z) Ys is the list that remains from the list Xs
%    when all leading copies of length L are removed and transfed to Z
transf(_,[],[],[]).
transf(L-_,[K-Y|Ys],[K-Y|Ys],[]) :- L \= K.
transf(L-_,[L-X|Xs],Ys,[L-X|Zs]) :- transf(L-X,Xs,Ys,Zs).
```

Solución del ejercicio 3.1

```
cbal_tree(0,void) :- !.
cbal_tree(N,tree(x,L,R)) :- N > 0,
        N0 is N - 1,
        N1 is N0//2, N2 is N0 - N1,
        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

```
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

a) `?- progenitor(sarah, isaac), mujer(sarah).`

b) `?- progenitor(X, isaac).`

c) `?- progenitor(X, isaac), mujer(X).`

d) `?- progenitor(abraham, X), varon(X).`

e) `?- mujer(X).`

f) Las respuestas generadas por `?- progenitor(terach,Y), progenitor(Y,N), mujer(N)` son correctas (asocia a `N` las dos nietas de terach, milcah y yiscah) pero facilita además los valores de los progenitores intermedios en la variable `Y`, información que no se requiere.

La respuesta correcta es `?- progenitor(terach,_Y), progenitor(_Y,N), mujer(N)`. Note que si en lugar de `_Y` usamos la variable anónima `_` además de `N=milcah` y `N=yiscah`, facilita la respuesta `N=sarah`, que no es nieta de terach. 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 terach es progenitor 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 sarah.

Trata de explicar porque repite las respuestas tres veces (Pista, pon variables distintas en lugar de las variables semianónimas).

g) `?- progenitor(X, abraham), progenitor(X, haran).`

h) `?- progenitor(_X, abraham), progenitor(_X, haran).`

Solución del ejercicio 4.2

a)
```
madre(X,Y) :- progenitor(X,Y), mujer(X).
```

b)
```
madre(X) :- madre(X,_).
```

c)
```
hija(X,Y) :- progenitor(Y,X), mujer(X).
```

d)
```
abuelo(X,Y) :- progenitor(X,Z), progenitor(Z,Y), varon(X).
```

e)
```
hermana(X,Y) :-
            progenitor(Z,X), progenitor(Z,Y), mujer(X), X\=Y.
```

f)
```
tia(X,Y) :- progenitor(Z,Y), hermana(X,Z).
```

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

h)
```
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

```
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

```
:- op(900, fy,  not).
:- op(910, yfx, and).
:- op(910, yfx, nand).
:- op(920, yfx, or).
:- op(920, yfx, nor).
:- op(930, yfx, impl).
:- op(930, yfx, equ).
:- op(930, yfx, xor).

table(A,B,Expr) :- bind(A), bind(B), do(A,B,Expr), fail.
[...]
```

Solución del ejercicio 5.3

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

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

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

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

writeExpr(Expr) :- Expr, !, write(true).
writeExpr(_) :- write(fail).
[...]
```

Solución del ejercicio 6.1

- Versión con doble recursión:

```
fibonacci(0,0).
fibonacci(1,1).
fibonacci(N,F) :-
        N > 1,
        N_1 is N-1,
        N_2 is N-2,
        fib(N_1, FN_1),
        fib(N_2, FN_2),
        F is FN_1 + FN_2.
```

- Versión con doble recursión por cola (con acumulador):

```
fibonacci(0,0).
fibonacci(1,1).
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).
```

Solución del ejercicio 7.1

```
1  :- use_module(library(actmod/actmod_dist)).
2  main([Exp]) :-
3       atom_to_term(Exp,Y),
4       X is Y,
5       display(X),nl.
```

Solución del ejercicio 7.2

```
1  :- module(main,_).
2  :- use_package(clpq).
3  :- use_module(library(actmod/actmod_dist)).
4  main(As) :-
5      atom_to_term(As,[Vs|Expr]),
6      clpq_meta(Expr),
7      display(Vs),nl.
```

a) `calc '[X,Y]' '3*X+5*Y .=. 2' '5*X+3*Y .=. -2'`

b) `calc '[A,B,C]'`
`'2*A+B-3*C .=. 7' '5*A-4*B+C .=. -19' 'A-B-4*C .=. 4'`