PROGRAMACIÓN DECLARATIVA
Relación 2

1. Podemos definir el factorial de la siguiente manera

```
f(0,1).
f(N,X):-
   N1 is N-1,
   f(N1,X1),
   X is X1 * N.
```

   - ¿Qué responde Prolog a la pregunta `?- f(3,X).`?
   - ¿Y a la pregunta `?- f(N,6).`?
   - ¿Y a la pregunta `?-f(3,X), f(N,X).`?
   - Define `suma/3` y `multiplica/3` basados en la aritmética del sucesor.
   - Define `fac_suc(N,X)` que se verifique si `X` es el factorial de `N` con la aritmética del sucesor. ¿Qué se obtiene con la pregunta `?- fac_suc(s(s(s(0))),X), fac_suc(N,X).`

2. En este ejercicio vamos a considerar el programa `familia.pl` del Tema 3.

   (a) Definir el predicado `hijo(X)` que se verifique si `X` figura en alguna lista de hijos.

   (b) Definir el predicado `existe(X)` que se verifique si X es una persona existente en la base de datos.

   (c) Haz una pregunta de forma que las sucesivas respuestas sean las listas de la forma `[nombre, apellido1, apellido2]` de todas las personas que existen.

   (d) Determinar todos los estudiantes nacidos antes de 1983.

   (e) Definir el predicado `fecha_de_nacimiento(X,Y)` de forma que si `X` es una persona, entonces `Y` es su fecha de nacimiento.

   (f) Buscar todos los hijos nacidos en 1982.

   (g) Definir el predicado `sueldo(X,Y)` que se verifique si el sueldo de la persona `X` es `Y`.

   (h) Buscar todas las personas nacidas antes de 1954 cuyo sueldo sea superior a 12.000.

   (i) Definir la relacion `total(L,Y)` de forma que si `L` es una lista de personas, entonces `Y` es la suma de los sueldos de las personas de la lista `L`.

   (j) Calcular los ingresos totales de cada familia.

3. En este ejercicio consideraremos el programa `automata.pl` del Tema 3

   (a) Definir la relación `acepta_acotada(S,L,N)` que se verifique si el autómata, a partir del estado `S`, acepta la lista `L` y la longitud de `L` es `N`.

   (b) Buscar las cadenas aceptadas a partir de `e1` con longitud 3.

(c) Definir la relación `acepta_acotada_2(S,L,N)` que se verifique si el autómata, a partir del estado `S`, acepta la lista `L` y la longitud de `L` es menor o igual que `N`.

4. [**Flach-94 p.52**] Considera el siguiente programa

```
p(a,X) :- q(X).
p(X,Y) :- r(Y,X).
r(b,a).
r(c,a).
q(b).
```

(a) Dibuja el árbol de resolución SLD para la pregunta `?- p(M,N).`

(b) Observa que la respuesta `{M/a,N/b}` se obtiene dos veces. ¿Podemos poner un corte de manera que esa respuesta sólo se obtenga una vez manteniendo la tercera respuesta?

5. [**Van Le-93 p. 115**] Considera el siguiente programa

```
admitido(Nombre):-
estudiante_universitario(Nombre),
examen_de_idiomas(Nombre,Nota),
Nota >= 5.

examen_de_idiomas(S,3):- elto(S,[jose,pedro,ana,alvaro,pablo]).
examen_de_idiomas(S,5):- elto(S,[lucia,antonio,andres]).
examen_de_idiomas(S,7):- elto(S,[agustin,carmen]).

estudiante_universitario(ana).
estudiante_universitario(lucia).
estudiante_universitario(antonio).
estudiante_universitario(agustin).

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

Este programa es ineficiente puesto que tras tener éxito `examen_de_idiomas(ana,3)` y fallar `3>=5`, intenta otro valor para el examen de `ana`

(a) ¿Mejora la eficiencia poniendo cortes en la definición de `elto`?

(b) Quita los cortes de `elto`. ¿Mejora la eficiencia poniendo cortes en la definición de `examen_de_idiomas` con las siguientes modificaciones? Prueba `?- admitido(N).` y `?- examen_de_idiomas(Alum,Calif).` e intenta obtener todas las soluciones.

- Modificación 1:

```
examen_de_idiomas(S,3) :- !,elto(S,[jose,pedro,ana,antonio,pablo]).
examen_de_idiomas(S,5) :- !,elto(S,[lucia,antonio,andres]).
examen_de_idiomas(S,7) :- elto(S,[agustin,carmen]).
```

- Modificación 2:
```
examen_de_idiomas(S,3) :- elto(S,[jose,pedro,ana,alvaro,pablo]),!.
examen_de_idiomas(S,5) :- elto(S,[lucia,antonio,andres]),!.
examen_de_idiomas(S,7) :- elto(S,[agustin,carmen]).
```

(c) Vuelve a eliminar los cortes de `examen_de_idiomas` y considera el programa con la siguiente modificación:

```
admitido(Nombre):-
estudiante_universitario(Nombre),
una_vez(examen_de_idiomas(Nombre,Nota)),
Nota >= 5.

una_vez(P) :- P,!.
```

Estudia las mejoras de esta versión respecto del programa original.

6. [**Van Le-93, p. 125**] Escribir como programa lógico las siguientes afirmaciones:

Si X no está fuera entonces está en casa
Susana está fuera
Juan es el marido de Susana

Usa el programa para responder a las preguntas

(a) ¿Está Juan en casa?
(b) ¿Hay alguien en casa?

7. Dado el programa

```
p(1).
p(2) :- corte.
p(3).

corte:- !.
```

dibujar los árboles de resolución SLD correspondiente a las preguntas

- `?- p(X).`
- `?- p(X), corte, p(Y).`
- `?- p(X), !, p(Y).`

8. [**Bratko-86 p. 135**] Define el predicado `separa(Numeros, Pos_C, Neg)` que separa la lista de números `Numeros` en dos listas: Una de positivos (y cero si lo hubiera), `Pos_C` y otra de negativos `Neg`. Propón dos soluciones, una sin corte y otra con corte.

9. [**Bratko-86 p. 138**] Define el predicado `unificable(Lista1,Term, Lista2)` donde `Lista2` es la lista de miembros de `Lista1` que unifican con `Term`, pero que no están instanciados por esa unificación. Por ejemplo:

```
?- unificable([X,b,t(Y)],t(a),L).
   L=[X,t(Y)]
```

Fíjate que `X` e `Y` permanecen sin instanciar en la salida. (Idea: Usa `\+ Term1 = Term2`. Si `Term1` y `Term2` unifican, entonces `\+ Term1 = Term2` falla y deja `Term1` y `Term2` sin instanciar).

10. [**Clocksin-94 p. 84**] Usar `conc` para responder a la siguiente cuestión: *Encontrar todas las listas* `L` *tal que al concatenar* `L` *con* `[c,d]` *nos devuelva* `[a,b,c,d]`.

    (a) Si obtener prefijos de esta forma fuera el único uso de `conc` en nuestro programa, ¿podríamos modificar su definición para mejorar la eficiencia?

    (b) ¿Serviria esa modificación para el uso general de `conc`?.

11. Definir el predicado `suma_pares(L,N)` que toma como dato de entrada la lista de números enteros `L` y devuelva la suma `N` de todos los números pares que aparezcan en `L`. Por ejemplo

```
?- suma_pares([2,3,4],N).
   N = 6 ;
   No

?- suma_pares([1,3,5,6,9,11,24],N).
   N = 30 ;
   No
```

12. Definir el predicado `exponente_de_dos(N,Exp)` que tome como entrada un número entero positivo `N` y devuelva el valor `Exp`, que corresponde al exponente de 2 en la descomposición de `N` como producto de factores primos. Por ejemplos

```
?- exponente_de_dos(24,Exp).
   Exp=3;
   No
?- exponente_de_dos(49,Exp).
   Exp=0;
   No
?- exponente_de_dos(32,Exp).
   Exp=5;
   No
```