Práctico 8: Predicados extralógicos y de 2do. orden.

Ejercicio 1

Implemente el predicado largo(L,N) de manera que funcione para cualquier instanciación.

Ejercicio 2

a) Implemente los siguientes predicados en Prolog sobre valores **reales** de manera que puedan invocarse para cualquier instanciación, siempre que la cantidad de argumentos no instanciados sea como máximo 1.

suma(*X*, *Y*, *Z*)	←	*Z es la suma entre X e Y*.
producto(*X*, *Y*, *Z*)	←	*Z es el producto entre X e Y*.
cuadrado(*X*, *Y*)	←	Y es el cuadrado de X. Notar que para la invocación (-X,+Y) pueden existir 0, 1 o 2 valores de X posibles, dependiendo de Y.
exponencial(*X*, *Y*, *Z*)	←	*Z es* X^Y.

b) Implemente el predicado suma(X, Y, Z) sobre valores **enteros** que funcione para las instanciaciones de la parte a), y además para la instanciación (-X, -Y, +Z) considerando que X e Y serán mayores o iguales que 0.

Ejercicio 3

Implemente el predicado univ(?T,?L) utilizando los predicados functor y arg. Debe funcionar por lo menos para las instanciaciones (+T,-L) y (-T,+L).

(El predicado univ de SWI-Prolog se nota =.. y su lógica es la de unificar un término estructurado del lado izquierdo con una lista de functor + argumentos del lado derecho. Por ejemplo f(1,2,3) =.. [f,1,2,3]).

Ejercicio 4

Considere la representación de matrices mediante functores en Prolog. Una matriz se representa como un functor *m* aplicado a una serie de filas, y cada fila es un functor *f* aplicado a una serie de celdas. Por ejemplo, la siguiente matriz de tamaño 2x3:

1	2	3
4	5	6

se representa como: m(f(1, 2, 3),f(4, 5, 6))

Implemente los siguientes predicados:

matriz(+F,+C,+V,-M)	←	M es una matriz de F filas y C columnas donde cada celda tiene el valor V
celda(+M,?I,?J,?V)	←	V es el valor de la celda (I,J) de la matriz M
nuevo_valor(+M,+I, +J,+V)	←	Se sustituye el valor de la celda (I,J) de la matriz M por V*
suma(+M,+N,?S)	←	S es la suma de las matrices M y N

* Notar que nuevo_valor no tiene argumento de salida. Se sugiere investigar el predicado extralógico set_arg/3 de SWI-Prolog.

Ejercicio 5

Sea el siguiente programa Prolog:

```
padre(juan, ana).
padre(juan, jose).
padre(juan, pedro).
padre(pedro, hector).
padre(pedro, gustavo).
padre(hector, maria).
```

Indique las respuestas que se obtienen para L con los siguientes objetivos:

i. `findall(X, padre(juan,X), L).`
ii. `findall(X, padre(Y,X), L).`
iii. `findall(X, (padre(juan,X) ; padre(pedro,X)), L).`
iv. `findall(X, (padre(juan,X), padre(X,Y)), L).`
v. `setof(X, padre(juan,X), L).`
vi. `setof(X, padre(Y,X), L).`
vii. `setof(X, (padre(juan,X) ; padre(pedro,X)), L).`
viii. `setof(X, (padre(juan,X), padre(X,Y)), L).`

Ejercicio 6

Utilizando predicados de segundo orden, implemente los siguientes predicados:

pares(+*L*,?*P*) ← P contiene los elementos pares de L.

mayores(+*L*,+*X*,?*M*) ← M contiene los elementos de L que son mayores que X.

union(+*C1*,+*C2*,?*C*) ← *C* es la unión de los conjuntos *C1* y *C2*.

interseccion(+*C1*,+*C2*,?*C*) ← *C* es la intersección de los conjuntos *C1* y *C2*.

diferencia(+*C1*,+*C2*,-*C*) ← *C* es igual a *C1*-*C2*.

adyacentes(+*N*,?*A*) ← A es la lista de nodos adyacentes al nodo *N* en un grafo definido mediante el predicado *arista*(*N1*, *N2*).

max_comun(+*L1*,+*L2*,?*L*) ← *L* es la sublista más larga común a *L1* y *L2*.

Ejercicio 7

Implemente los siguientes predicados de segundo orden. Asuma que los argumentos U, B y T contendrán predicados unarios (por ejemplo par/1), binarios (por ejemplo doble/2) o ternarios (por ejemplo suma/3) respectivamente.

any(+*L*,+*U*) ← Algún elemento de L cumple la propiedad U.

all(+*L*,+*U*) ← Todos los elementos de L cumplen la propiedad U.

map(+*L*,+*B*,*?L2*) ← L2 es el resultado de aplicar la función B a todos los elementos de L.

combine(+*L1*,+*L2*,+*T*, *?L3*) ← L3 es el resultado de aplicar el operador T a elementos en las mismas posiciones de L1 y L2.

fold(+*L*,+*T*,*?F*) ← F es el resultado de realizar un *fold* sobre la lista L con el operador T. Por ejemplo, si T fuera la suma la operación sería:
$F = L_1 + L_2 + \ldots + L_{n-1} + L_n$

Ejercicio 8 [prueba 03]

Sea el siguiente programa Prolog:

```
between(M,N,M)  :- M=<N.
between(M,N,K)  :-
        M<N,
        M2 is M+1,
        between(M2,N,K).
par(N) :- N mod 2 =:= 0.
par_menor(N,M)  :-
        between(1,N,M),
        par(M).
todos_q(Q,Xs)   :-
        findall(X,(append(Q,[X],TL), T=..TL, call(T)),Xs).
```

a) Dé los valores de *Xs* que son solución de la consulta
?- todos_q([par_menor,9], Xs).

b) Para las siguientes variantes con *cut* del programa anterior, indique los valores de *Xs* que son solución de la consulta *?- todos_q([par_menor,9], Xs).* Justifique sus respuestas.

i.

```
between(M,N,M)  :- M=<N.
between(M,N,K)  :-
        M<N,
        M2 is M+1,
        between(M2,N,K).
par(N) :- N mod 2 =:= 0.
par_menor(N,M)  :-
        between(1,N,M),
        !,
        par(M).
todos_q(Q,Xs)   :-
        findall(X,(append(Q,[X],TL), T=..TL, call(T)),Xs).
```

ii.

```
between(M,N,M)  :- M=<N.
between(M,N,K)  :-
        M<N,
        M2 is M+1,
        between(M2,N,K).
par(N) :- N mod 2 =:= 0.
par_menor(N,M)  :-
        !,
        between(1,N,M),
        par(M).
todos_q(Q,Xs)   :-
        findall(X,(append(Q,[X],TL), T=..TL, call(T)),Xs).
```

iii.

```
between(M,N,M)  :- M=<N.
between(M,N,K)  :-
      M<N,
      M2 is M+1,
      between(M2,N,K).
par(N) :- N mod 2 =:= 0.
par_menor(N,M)  :-
      between(1,N,M),
      par(M).
todos_q(Q,Xs)   :-
      findall(X,(append(Q,[X],TL), T=..TL, call(T),!),Xs).
```

Ejercicio 9 [prueba 15]

Considere un laberinto modelado mediante una matriz de enteros en Prolog. Las celdas de la matriz pueden valer 0 cuando se permite pasar o 1 cuando no se permite (es una pared). Por ejemplo, la siguiente matriz representa un laberinto:

0 1 0 0 0 0
0 1 0 1 1 0
0 0 0 0 0 0
0 1 0 1 0 1

La representación interna de la matriz no es conocida, se cuenta solamente con el siguiente predicado para su manejo:
`celda(+L, +I, +J, ?V)` <- V es el valor de la celda en la fila I y columna J de la matriz L. Si la fila o la columna no existen, el predicado falla.

Implementar los siguientes predicados:

a) `adyacente_valido(+L,+I1,+J1,?I2,?J2)` <- `(I1,J1)` e `(I2,J2)` son celdas adyacentes en L y además se puede pasar de una a la otra (las dos valen 0). Los movimientos válidos son horizontal y verticalmente, no hay movimiento en diagonal.

b) `camino(+L, +I1, +I2, ?J1, ?J2, ?C)` <- `C` es un camino para ir de la celda (I1,J1) a la celda (I2,J2) de L. C es una lista de elementos de la forma celda(X,Y).

c) `camino_mas_corto(+L, +I1, +I2, ?J1, ?J2, ?C)` <- `C` es el camino más corto (o uno de los caminos más cortos si hay varios de la misma distancia) para ir de la celda `(I1,J1)` a la celda `(I2,J2)` de L.

d) `alcanzables(+L,+I,+J,+N,?C)` <- `C` es una lista con todas las celdas de L para las que existe un camino desde la celda `(I,J)` que pase como máximo por `N` celdas. La lista `C` puede tener elementos repetidos.

Ejercicio 10

Implemente el predicado **findall** utilizando los predicados **assert** y **retract**.

Ejercicio 11

Implemente un shell de Prolog en Prolog, que permita generar un archivo con el registro de las consultas y respuestas dadas.

shell(Log) ← *Shell* inicia un shell de Prolog que devuelve todas las respuestas a una consulta sin dar punto y coma. *Log* puede ser el átomo *log*, o *nolog*. Si es *log*, todas las entradas y respuestas deben ser registradas en el archivo 'shell.log'.