**1.**Implementar un predicado Prolog de sintaxis **duplicar(Lista, Resultado)** tal que

Resultado es la lista resultante de duplicar en la lista Lista cada uno de sus elementos.

**Ejemplo:** La respuesta a la pregunta **duplicar([a,b,c,c,d],X).** es **X = [a,a,b,b,c,c,c,c,d,d].**

**duplicar( [ ] , [ ] ).**

**duplicar( [ Car | Cdr ] , [ Car , Car | R ] ) :- duplicar( Cdr , R ).**

**2.**Implementar un predicado **comp(Subconj, Conj, Comp)** que se verifique sii Comp es el

complementario del subconjunto Subconj en el conjunto Conj.

NOTA: Podemos suponer que los argumentos son listas sin elementos repetidos.

**Ejemplo:** Una respuesta a la pregunta **comp([3,1], [1,3,2,6], X)** es **X = [2, 6]** o cualquier otra

permutación de X.

**comp( \_ , [ ] , [ ] ).**

**comp( Conj , [ Car | Cdr ] , R ) :- miembro( Car , Conj ) , ! , comp( Conj , Cdr , R ).**

**comp( Conj , [ Car | Cdr ] , [ Car | R ] ) :- comp( Conj , Cdr , R ).**

**3.**Implementar un predicado Prolog de sintaxis **get-asoc(Llave,ListAsoc,Valor)** tal que Valor

es el resultado de recuperar en la lista de asociación ListAsoc el valor asociado a la clave Llave. Suponemos que una lista de asociación es un conjunto de pares [Llave, Valor] agrupados en una lista, donde las llaves no se repiten.

**Ejemplo:** La respuesta a la pregunta **get-asoc(b,[[a,1],[b,2],[c,3]], X)** es **X=2.**

**get-asoc( \_ , [ ] , [ ] ).**

**get-asoc( Llave , [ [ Llave , Valor ] | \_ ] , Valor ) :- !.**

**get-asoc( Llave , [ \_ | Cdr ] , Valor ) :- get-asoc( Llave , Cdr , Valor ).**

**4.**Implementar un predicado Prolog de sintaxis **eliminar(Lista,Posición,Resultado)** tal que Resultado es la lista obtenida a partir de la original Lista, una vez eliminado el elemento en la posición Posición.

**Ejemplo:** La respuesta a **eliminar([a,b,c,d,e,f,g,h,i,k],3,X).** es **X = [a,b,d,e,g,h,k]**.

**eliminar( L1 , Pos , L2 ) :- eliminar( L1 , Pos , L2 , Pos ).**

**eliminar( [ ] , \_ , [ ] , \_ ).**

**eliminar( [ \_ | Cdr ] , Pos , R , 1 ) :- eliminar( Cdr , Pos , R , Pos ).**

**eliminar( [ Car | Cdr ] , Pos , [ Car | R ] , P ) :- P > 1 ,P1 is P - 1, eliminar(Cdr , Pos , R , P1).**

**5.**Implementar un predicado Prolog de sintaxis **borrar-repes(Lista,Resultado)** tal que

Resultado es la lista resultante de borrar de la lista Lista aquellos elementos que se repiten.

**Ejemplo:** La respuesta a la pregunta **borrar-repes([1,2,[3,4],2,3,4,1], X).** es **X=[[3,4],3,4].**

**borrar-repes( Lista , R ) :- repes( Lista , Repes ) , comp( Repes , Lista , R ).**

**repes( [ ] , [ ] ).**

**repes( [Car | Cdr] , [Car | R] ) :- miembro( Car , Cdr ) , ! , repes( Cdr , R ).**

**repes( [Car | Cdr] , R ) :- repes( Cdr , R ).**

**miembro( Elem , [ Elem | \_ ] ).**

**miembro( Elem , [ \_ | Cdr ] ) :- miembro( Elem , Cdr ).**

**6.**Implementar un predicado Prolog de sintaxis **insert ceros(Lista,Resultado)** tal que

Resultado es la lista resultante de intercalar ceros entre los elementos de la lista Lista.

**Ejemplo:** La respuesta a la pregunta **insert ceros([1,2,[3,4],5], X) es X=[1,0,2,0,[3,4],0,5,0].**

**insert-ceros( [ ] , [ ] ).**

**insert-ceros( [Car | Cdr] , [Car , 0 | R] ) :- insert-ceros( Cdr , R ).**

**7.**Implementar un predicado Prolog de sintaxis **borrar-unicos(Lista,Resultado)** tal que

Resultado es la lista resultante de borrar de la lista Lista aquellos elementos que no se repiten.

**Ejemplo:** La respuesta a la pregunta **borrar-unicos([1,2,[3,4],2,3,4,1], X).** es **X=[[3,4],3,4].**

**borrar-unicos( Lista , R ) :- borrar-repes( Lista , Unicos ), comp( Unicos , Lista , R ).**

**8.**Implementar un predicado Prolog de sintaxis **binario(Funcion)** que sea cierto si la estructura

Funcion se corresponde a un árbol binario.

**Ejemplo:** La respuesta a la pregunta **binario(arbol(a,arbol(b,nil,nil),nil)).** es cierto, lo mismo

para la pregunta **binario(a,nil).**

NOTA: Representaremos un árbol vacío mediante nil, y los términos no vacíos mediante la función arbol(Raiz,Izqd,Der), donde Raiz es la etiqueta del nodo raíz, mientras Izqd es el hijo izquierdo y Der es el derecho.

**binario( arbol ( Atom1 , Atom2 ) ) :- atom( Atom1 ) , atom( Atom2 ).**

**binario( arbol ( Atom , R ) ) :- atom ( Atom ) , binario ( R ).**

**binario( arbol ( L , Atom ) ) :- binario ( L ) , atom ( Atom ).**

**binario( arbol ( L , R ) ) :- binario ( L ) , binario ( R ).**

**9.**Implementar un predicado Prolog de sintaxis **subarbol(Subarbol,Arbol\_binario)** que sea cierto si y sólo si Subarbol es un sub-árbol del árbol binario Arbol\_binario.

NOTA: Supondremos que un árbol binario se implementa usando la función arbol(Izqd,Der), donde Izqd es el subárbol binario hijo a la izquierda y Der es el subárbol binario hijo a la derecha.

**Ejemplo:** La respuesta a **subarbol(arbol(a,b),arbol(arbol(c,d),arbol(e,arbol(a,b))))** es cierto.

**subarbol( arbol ( L , R ) , arbol ( L , R ) ).**

**subarbol( S , arbol ( S , \_ ) ) :- !.**

**subarbol( S , arbol ( \_ , S ) ) :- !.**

**subarbol( S , arbol ( L , R ) ) :- subarbol( S , L ) ; subarbol( S , R ).**