Ejercicio 1: El predicado pertenece comprueba si el elemento x pertenece a una lista.

```
pertenece(X,[X|_]).
pertenece(X,[_|L2]):- pertenece(X,L2).
```

Lo hemos realizando tomando como case base que el elemento que queremos comprobar es el primero de la lista, y como caso recursivo volver a evaluar con el resto de la lista.

Ejercicio 2: El predicado invierte comprueba si el segundo argumento que es una lista es la inversa que el primer argumento, que también es una lista.

```
invierte([],[]).
invierte([H|T],L):- invierte(T,R), concatena(R,[H],L).
```

Hemos realizado este predicado tomando como caso base dos listas vacías, que evidentemente se cumple que sean la inversa la una de la otra.

En el caso recursivo llamamos al propio predicado con el resto de la lista inicial, y de segundo argumento el resultado de concatenar el primer elemento de la lista original con la lista invertida.

Ejercicio 3:

Se fija que el nodo raíz es el primer elemento de la lista en la declaración. Posteriormente se explora recursivamente el nodo de la izquierda a la vez que se va concatenando el nodo actual. Cuando se llega al último nodo de la izquierda (hijo de la izquierda nil), se empieza a explorar el nodo de la derecha y cuando se va deshaciendo la recursión, también se exploran los nodos de la derecha (en ambos casos como si fueran subárboles realizando el algoritmo desde el principio).

```
preorder(nil, []).
preorder(tree(INFO, LEFT, RIGHT), [INFO|L]) :- preorder(LEFT, LL),
                                                concatena(LL, RL, L),
                                                preorder(RIGHT, RL).
```

Se utiliza recursión hasta llegar a la hoja de más a la izquierda, una vez ahí se añade el nodo actual a la lista. Cuando se empieza a deshacer la recursión se añade a la lista el nodo actual (padre del anterior) y se explora el subárbol de la derecha aplicando el mismo algoritmo desde el principio.

```
inorder(nil, []).
inorder(tree(INFO, LEFT, RIGHT), L):- inorder(LEFT,LL),
                                      inorder(RIGHT, RL),
                                      concatena(LL, [INFO|RL],L).
```

Se utiliza recursión para llegar al nodo de más a la izquierda y se concatena con su hermano. Mientras se deshace la recursión de va repitiendo la operación con los padres.

```
postorder(nil, []).
postorder(tree(INFO, LEFT, RIGHT), L):- postorder(LEFT ,LL),
                                        postorder(RIGHT, RL),
                                        concatena(LL, RL,R1),
                                        concatena(R1, [INFO], L).
```

Ejercicio 4: El predicado inserta el elemento X en la lista L en la posición Pos

```
insertar(X, L, 1, [X|L]).
insertar(X,[H|L],Pos,[H|R]):- Pos > 1, !, Pos1 is Pos - 1,
insertar(X,L,Pos1,R).
```

Ejercicio 5: Este predicado se encarga de extraer un elemento dado de una lista, y en el tercer argumento va la lista resultado de la extracción

```
extract(L1,X,L2):- concatena(L4,L5,L1),
                   concatena([X],L3,L5),
                   concatena(L4,L3,L2).
```

Lo hemos realizado haciendo uso del predicado concatena.

Ejercicio 6: Implementamos la tabla de preferencias que se nos da en el enunciado

```
man_pref(juan, [maria, carmen, pilar]).
man_pref(pedro, [carmen, maria, pilar]).
man_pref(mario, [maria, carmen, pilar]).

woman_pref(maria, [juan, pedro, mario]).
woman_pref(carmen, [pedro, juan, mario]).
woman_pref(pilar, [juan, pedro, mario]).
```

Ejercicio 7: Comprueba si un matrimonio es estable o no basándose en las tablas de referencias anteriores y en si prefiere más a su cónyuge o a otro.

```
pos(X,[X|_], 1).
pos(X,[_|R],N):- pos(X,R,M), N is M + 1.
```

```
unstable(_-_, []) :- not(!).
unstable(M1-W1, [M1-W1]) :- not(!).
unstable(M1-W1, [C|R]):- man_pref(M1, WL1),
                         woman_pref(W1, ML1),
                         pertenece(M2-_, [C]),
                         pertenece(_-W2, [C]),
                         pos(M2, ML1, Pos1),
                         pos(M1, ML1, Pos2),
                         pos(W2,WL1, Pos3),
                         pos(W1,WL1, Pos4),
                         Pos1<Pos2,
                         Pos3<Pos4,
                         unstable(M1-W1, R).
```

Ejercicio 8:

Hemos implementado el caso base para crear matrimonios estables según el hombre dado, la mujer dada y la lista de matrimonios. Comprueba si el posible matrimonio seria estable o no. Si es estable lo concatena en la lista MarriagesOut.

```
smp(FreeMen, FreeWoman, MarriagesIn, MarriagesOut):-
    not(unstable(FreeMen-FreeWoman, MarriagesIn)),
    concatena(MarriagesIn, [FreeMen-FreeWoman], MarriagesOut).
```