

# Paradigmas de Programación

## Programación Lógica

### Guía de Ejercicios N° 2

## Índice de ejercicios

<b>Parte I</b>	<b>2</b>
Ejercicio 1. Predicados con listas . . . . .	2
concatenar/3 . . . . .	2
prefijo/2 . . . . .	2
sufijo/2 . . . . .	2
sublista/2 . . . . .	2
ultimo/2 . . . . .	2
miembro/2 . . . . .	2
adyacente/3 . . . . .	3
selecciona/3 . . . . .	3
reversa/2 . . . . .	3
permuta/2 . . . . .	3
Ejercicio 2. Miembro en listas anidadas . . . . .	3
Ejercicio 3. Lista de elementos atómicos . . . . .	3
Ejercicio 4. Subsecuencia . . . . .	4
Ejercicio 5. Predicados con árboles binarios . . . . .	4
arbolBinario/1 . . . . .	4
miembroArbol/2 . . . . .	4
listaHojas/2 . . . . .	4
preorden/2 . . . . .	5
inorden/2 . . . . .	5
posorden/2 . . . . .	5
<b>Parte II</b>	<b>5</b>
Ejercicio 6. Predicados relacionales y/o aritméticos con listas . . . . .	5
longitud/2 . . . . .	5
maximo/2 . . . . .	5
minimo/2 . . . . .	5
enesimo/3 . . . . .	5
sumaLista/2 . . . . .	5
sinDuplicados/2 . . . . .	5
ordenada/2 . . . . .	6
reemplaza/4 . . . . .	6
eliminar/3 . . . . .	6
Ejercicio 7. Predicados relaciones y/o aritméticos con árboles binarios . . . . .	6
profundidad/2 . . . . .	6
sumaNodos/2 . . . . .	6
cuentaHojas/2 . . . . .	6

## Parte I

### Ejercicio 1

#### Predicados con listas

Definir los siguientes predicados con listas. Tener en cuenta que serán útiles como predicados auxiliares para resolver ejercicios de las guías prácticas.

- a) `concatenar(Xs, Ys, XsYs)`: evalúa verdadero si `XsYs` es la concatenación de las listas `Xs` e `Ys`.

Ejemplos:

```
?- concatenar([1, 2], [a, b, c], [1, 2, a, b, c]).
```

```
true1
```

```
?- concatenar([1, 2], [a, b, c], L).
```

```
L = [1, 2, a, b, c]
```

```
?- concatenar(L, [a, b, c], [1, 2, a, b, c]).
```

```
L = [1, 2]
```

```
?- concatenar([1, 2], L, [1, 2, a, b, c]).
```

```
L = [a, b, c]
```

```
?- concatenar(L1, L2, [1, 2, a, b, c]).
```

```
L1 = [], L2 = [1, 2, a, b, c] ;
```

```
L1 = [1], L2 = [2, a, b, c] ;
```

```
L1 = [1, 2], L2 = [a, b, c] ;
```

```
L1 = [1, 2, a], L2 = [b, c] ;
```

```
L1 = [1, 2, a, b], L2 = [c] ;
```

```
L1 = [1, 2, a, b, c], L2 = []
```

- b) `prefijo(Prefijo, Lista)`: evalúa verdadero si `Prefijo` es prefijo de `Lista`.

Ejemplos:

```
?- prefijo([b1, c1], [a1, b1, c1, d1]).
```

```
false
```

```
?- prefijo(P, [a1, b1, c1, d1]).
```

```
P = [] ;
```

```
P = [a1] ;
```

```
P = [a1, b1] ...
```

- c) `sufijo(Sufijo, Lista)`: evalúa verdadero si `Sufijo` es sufijo de `Lista`.

Ejemplos:

```
?- sufijo([b1, c1], [a1, b1, c1, d1]).
```

```
false
```

```
?- sufijo(P, [a1, b1, c1, d1]).
```

```
P = [a1, b1, c1, d1] ;
```

```
P = [b1, c1, d1] ;
```

```
P = [c1, d1] ...
```

- d) `sublista(Sub, Lista)`: evalúa verdadero si `Sub` es sublista de `Lista`. Puede considerar el uso de los predicados definidos anteriormente.

Ejemplos:

```
?- sublista([1, 2, 3, 5], [1, 1, 2, 3, 5, 8, 13, 21]).
```

```
true
```

```
?- sublista([2, 5], [1, 1, 2, 3, 5, 8, 13, 21]).
```

```
false
```

```
?- sublista([21], [1, 1, 2, 3, 5, 8, 13, 21]).
```

```
true
```

- e) `ultimo(Elemento, Lista)`: se satisface si `Elemento` es el último elemento de `Lista`.

- f) `miembro(Elemento, Lista)`: se satisface si `Elemento` es un elemento de `Lista`.

Ejemplos:

<sup>1</sup>Para consultas básicas ("ground") exitosas la salida puede variar según la configuración del intérprete Prolog.

```
?- miembro("dos", ["uno", "dos", "tres", "nueve"]).
true
?- miembro(Elem, ["uno", "dos", "tres", "nueve"]).
Elem = "uno" ;
Elem = "dos" ...
```

g) `adyacente(X, Y, Lista)`: evalúa verdadero si `X` e `Y` son elementos adyacentes en `Lista`.

h) `selecciona(Lista, Elemento, ListaR)`: se satisface si `ListaR` es la resultante de eliminar una ocurrencia de `Elemento` en `Lista`.

Ejemplos:

```
?- selecciona([a, b, c, d, a], E, L).
E = a, L = [b, c, d, a] ;
E = b, L = [a, c, d, a] ...
```

i) `reversa(Lista, LInvertida)`: evalúa verdadero si `LInvertida` es la resultante de invertir `Lista`.

Ejemplos:

```
?- reversa([l, o, g, i, c, a], T).
T = [a, c, i, g, o, l]
```

j) `permute(Lista, Permuta)`: evalúa verdadero si `Permuta` corresponde a una permutación de `Lista`. Si el tamaño de `Lista` es  $n$ , la cantidad de soluciones posibles será  $n!$

Ejemplos:

```
?- permute([a, b, c], P).
P = [a, b, c] ;
P = [a, c, b] ;
P = [b, a, c] ;
P = [b, c, a] ;
P = [c, a, b] ;
P = [c, b, a]
```

## Ejercicio 2

### Miembro en listas anidadas

Definir el predicado `ocurre(Elemento, Lista)` que deberá evaluar verdadero cuando `Elemento` ocurra en cualquier lugar de `Lista`. Tener en cuenta que `Lista` puede contener listas (cualquier nivel de anidamiento).

Ejemplos:

```
?- ocurre(2, [1, [a, b], [d, [a, 2]], 6]).
true
?- ocurre(E, [1, [a, b], [d, [a, 2]], 6]).
E = 1 ;
E = [a, b] ...
```

## Ejercicio 3

### Lista de elementos atómicos

Escribir un programa que permita obtener la lista de átomos de una lista dada, la cual a su vez, puede contener listas sin límite de anidamientos.

Ejemplos:

```
?- listaAtomos([1, [[2, a], 3], [4]], [1, 2, a, 3, 4]).
true
?- listaAtomos([[5, s], [p, q]], r, [2]), X.
X = [5, s, p, q, r, 2]
```

## Ejercicio 4

### Subsecuencia

Dada una secuencia de números, una subsecuencia se obtiene seleccionando elementos, no necesariamente adyacentes, de la secuencia original conservando el orden de los mismos. Por ejemplo:

$$S = (2\ 4\ 8\ 3\ 100\ 93\ 50\ 55\ 89\ 3\ 1)$$

Las siguientes son subsecuencias de S: (2 8 3 100), (4 3 50 1), (8 3 100 89 3 1), ...

Definir el predicado `subSecuencia(Secuencia, Sub)` que evalúa verdadero si `Sub` es una subsecuencia de `Secuencia`.

Ejemplos:

```
?- subSecuencia([2, 4, 8, 3, 4], [4, 2]).
```

```
false
```

```
?- subSecuencia([2, 4, 8, 3, 4], Sub).
```

```
Sub = [] ;
```

```
Sub = [4] ;
```

```
...
```

```
Sub = [8, 3] ;
```

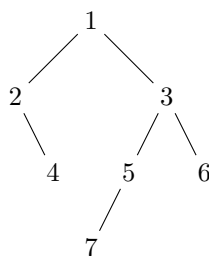
```
...
```

```
Sub = [2, 4, 8, 4] ...
```

## Ejercicio 5

### Predicados con árboles binarios

Un árbol binario, es decir, que contiene a lo sumo dos descendientes (o hijos), puede ser representado por la función `arbol(Raiz, Arbol1, Arbol2)`. Los argumentos corresponden a la raíz (**Raiz**) y los descendientes `Arbol1` y `Arbol2` (también denominados izquierdo y derecho, respectivamente). Tener en cuenta que en caso de no poseer un descendiente se usará `nil` en su lugar. Ejemplo de árbol binario y su representación en la variable `Arbol`:



```
Arbol = arbol(1,
    arbol(2, nil, arbol(4, nil, nil)),
    arbol(3, arbol(5, arbol(7, nil, nil), nil),
    arbol(6, nil, nil)))
```

Definir los predicados básicos solicitados a continuación.

a) `arbolBinario(Arbol)`: evalúa verdadero si el argumento es un árbol binario.

b) `miembroArbol(Valor, Arbol)`: evalúa verdadero si `Valor` se encuentra almacenado en un nodo del árbol binario `Arbol`.

Ejemplos:

```
?- miembroArbol(9, Arbol).
```

```
false
```

```
?- miembroArbol(X, Arbol).
```

```
X = 1 ;
```

```
X = 2 ...
```

c) `listaHojas(Arbol, Hojas)`: se satisface si la lista `Hojas` contiene los valores almacenados en las hojas de `Arbol`.

Ejemplos:

```
?- listaHojas(arbol(9, nil, nil), [9]).
```

```
true
```

```
?- listaHojas(Arbol, L).
```

```
L = [4, 7, 6]
```

- ✓ d) `preorden(Arbol, Lista)`: evalúa verdadero si el segundo argumento es una lista que contiene los valores de los nodos del árbol binario `Arbol` según recorrido preorden.  
Ejemplo:  
`?- preorden(Arbol, L).`  
`L = [1, 2, 4, 3, 5, 7, 6]`
- ✓ e) `inorden(Arbol, Lista)`: ídem anterior pero según recorrido inorden.
- ✓ f) `posorden(Arbol, Lista)`: ídem anterior pero según recorrido posorden.

## Parte II

### Ejercicio 6

#### Predicados relacionales y/o aritméticos con listas

Definir los siguientes predicados:

- ✓ a) `longitud(Lista, N)`: evalúa verdadero si `N` es la longitud de `Lista`.  
Ejemplos:  
`?- longitud([t0, t1, t2, t3, t4, t5], N).`  
`N = 6`  
`?- longitud(["declarative programming"], 2).`  
`false`
- ✓ b) `maximo(Lista, Max)`: se satisface si `Max` es el valor máximo de `Lista`.  
Ejemplos:  
`?- maximo([2, 0, 9, 1, 5], N).`  
`N = 9`  
`?- maximo([4, 5, 10, 3, 10, 2, 10, 0], 10).`  
`true`
- ✓ c) `minimo(Lista, Min)`: ídem predicado anterior pero con el valor mínimo.
- ✓ d) `enesimo(Lista, N, Elemento)`: evalúa verdadero si `Elemento` se encuentra en la posición `N` de `Lista`.  
Ejemplos:  
`?- enesimo([a, b, c, d], 4, d).`  
`true`  
`?- enesimo([a, b, c, d], 3, E).`  
`E = c`  
`?- enesimo([a, b, c, d], N, E).`  
`N = 1, E = a ;`  
`N = 2, E = b ...`
- ✓ e) `sumaLista(Lista, Suma)`: se satisface si `Suma` es la suma de los elementos de `Lista`.  
Ejemplos:  
`?- sumaLista([1, 2, 4, 8, 16, 32, 64], N).`  
`N = 127`  
`?- sumaLista([], 0).`  
`true`
- ✓ f) `sinDuplicados(Lista, LSinDup)`: se satisface si `LSinDup` unifica con la lista resultante de quitar los elementos repetidos de `Lista`.  
Ejemplos:  
`?- sinDuplicados(["00", "10", "00", "11", "10", "01"], L).`  
`L = ["00", "10", "11", "01"]`  
`?- sinDuplicados([1, 2, 4, 8, 16], L).`  
`L = [1, 2, 4, 8, 16]`

- g) `ordenada(Lista, LOrdenada)`: evalúa verdadero si `LOrdenada` contiene los mismos elementos de `Lista` pero ordenados ascendentemente.  
Ejemplos:  
`?- ordenada([10, 2, 5, -1, 4], [-1, 2, 4, 5, 10]).`  
`true`  
`?- ordenada([5, 0, 1, 2, 0], L).`  
`L = [0, 0, 1, 2, 5]`
- h) `reemplaza(Lista1, X, Y, Lista2)`: es exitoso si `Lista2` es la resultante de reemplazar todas las ocurrencias del elemento `X` por el elemento `Y` en `Lista1`.  
Ejemplos:  
`?- reemplaza([a, b, c, d, b, b, c, 1], b, 9, L).`  
`L = [a, 9, c, d, 9, 9, c, 1]`  
`?- reemplaza([a, b, c, d, b, b, c, 1], z, 0, L).`  
`L = [a, b, c, d, b, b, c, 1]`  
`?- reemplaza([a, z], 0, a, [a, z]).`  
`true`
- i) `elimina(Lista, Elemento, ListaR)`: evalúa verdadero si el tercer argumento es la lista resultante de eliminar todas las ocurrencias de `Elemento` en `Lista`.  
Ejemplos:  
`?- elimina([a, b, c], c, [a, b]).`  
`true`  
`?- elimina([a, b, c, b, d], b, L).`  
`L = [a, c, d]`

## Ejercicio 7

### Predicados relaciones y/o aritméticos con árboles binarios

Definir los siguiente predicados:

- a) `profundidad(Arbol, N)`: evalúa verdadero si `N` es la profundidad de `Arbol`.  
Ejemplos:  
`?- profundidad(Arbol2, N).`  
`N = 3`  
`?- profundidad(arbol(9, nil, nil), 0).`  
`true`
- b) `sumaNodos(Arbol, Suma)`: evalúa verdadero si el segundo argumento es la suma de los valores almacenados en los nodos de `Arbol`.  
Ejemplos:  
`?- sumaNodos(Arbol2, N).`  
`N = 28`  
`?- sumaNodos(arbol(0, nil, nil), 0).`  
`true`
- c) `cantidadHojas(Arbol, Suma)`: evalúa verdadero si el segundo argumento corresponde a la cantidad de hojas de `Arbol`.  
Ejemplos:  
`?- cantidadHojas(Arbol2, N).`  
`N = 3`  
`?- cantidadHojas(arbol(4, nil, nil), 1).`  
`true`

---

<sup>2</sup>Corresponde a la variable del ejercicio 5.