Estructuras de Datos en PROLOG: LISTAS

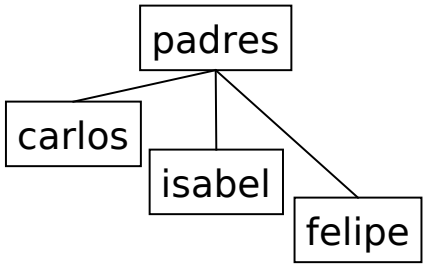
UNIVERSIDAD DE VALLADOLID

Departamento de Informática

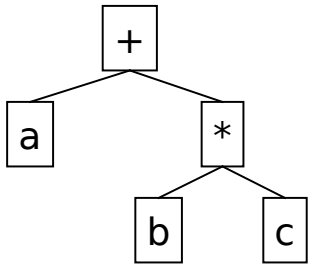
Estructuras y árboles

- Una estructura de datos en PROLOG se representa mediante un árbol

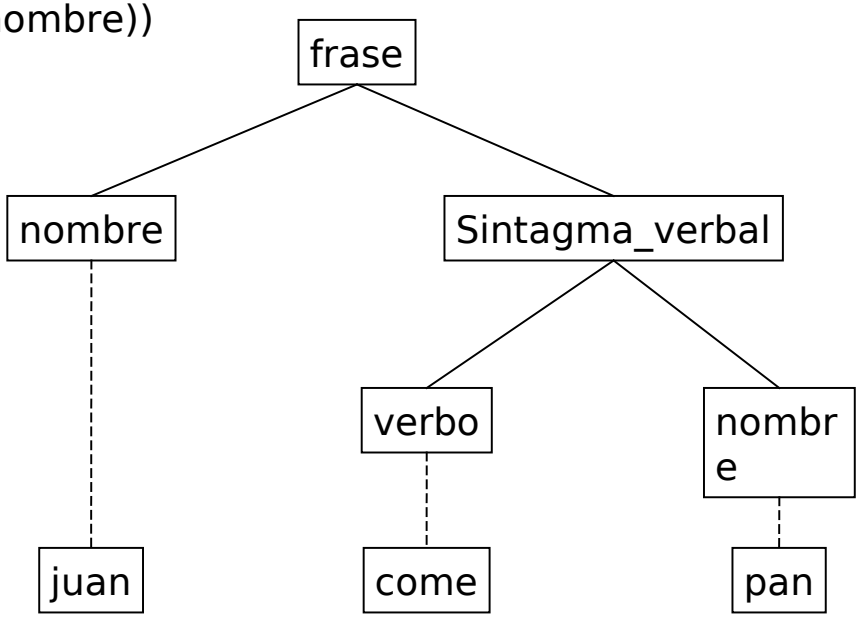
padres(hijo, madre, padre)
padres(carlos, isabel, felipe)



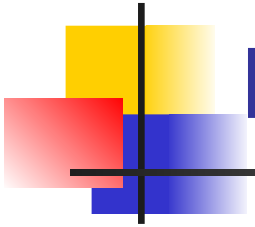
a+b*c



frase(nombre, sintagma_verbal(verbo, nombre))

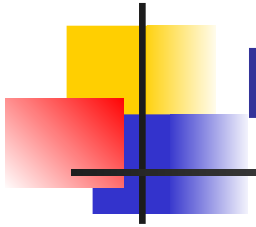


frase(nombre(juan), sintagma_verbal(verbo(come), nombre(pan)))



Listas (I)

- Lista es una **secuencia ordenada** de términos (constantes, variables, estructuras, e, incluso, otras lista).
- La lista es **ampliamente utilizada**: análisis sintáctico, gramáticas, diagramas de flujo, grafos, etc.
- Manejo específico de listas -> **LISP**.
- La lista es un caso particular de estructura en PROLOG => **recursividad** en la definición.



Listas (II)

- Tipo particular de árbol.
  - [Cabeza|Cola]:
    - La cabeza es un sólo elemento
    - La cola es el resto que, a su vez, es otra lista
  - El final de la lista es una lista vacía: “[]”
- Representación habitual de las listas:
  - [a,b,c]
  - [los, hombres, [van, a, pescar]]
- El acceso es secuencial:
  - Se va instanciando progresivamente:
    - Lista = [Cabeza1|Cola1],
    - Cola1 = [Cabeza2|Cola2]
    - Etc. hasta llegar a un Cabeza késima igual al elemento buscado

Pertenencia (I)

- Saber si un objeto pertenece a lista.

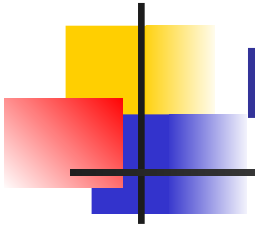
  [carlos_i, felipe_ii, felipe_iii, felipe_iv, carlos_ii]

- Construir el predicado “miembro”:
  - Caso base: `miembro(X, [Y|_]):- X=Y.`
  - Recursividad: `miembro(X, [_|Y]) :- miembro(X,Y).`

- Verifica la coincidencia con la cola y de forma recursiva va operando sobre otra lista progresivamente más pequeña:

  ?- miembro(felipe_ii, [carlos_i, felipe_ii, felipe_iii, felipe_iv, carlos_ii]).

  ?- miembro(X, [carlos_i, felipe_ii, felipe_iii, felipe_iv, carlos_ii]).



Práctica: alterar frase

- Diálogo:
  - Usuario: tu eres un ordenador
  - Prolog: yo no soy un ordenador
  - Usuario: hablas francés
  - Prolog: no_hablo alemán (“_” simula a “,”)
- Reglas:
  - Aceptar frase en formato lista:

    [tu, eres, un ordenador]
  - Cambiar cada *eres* por un *no soy.*
  - Cambiar cada *hablas* por un *no_hablo.*
  - Cambiar cada *francés* por un *alemán.*

Solución: alterar frase

```
cambiar(tu, yo).
cambiar(eres, [no, soy]).
cambiar(hablas, [no_,
    hablo]).
cambiar(frances,
    aleman).
cambiar(X,X).


alterar([], []).
alterar([H|T], [X|Y]):-
    cambiar(H,X),
    alterar(T,Y).
```

```
?- alterar([tu, hablas, frances],
    X).

X = [yo, [no_, hablo], aleman] ;

X = [yo, [no_, hablo], frances] ;
X = [yo, hablas, aleman] ;
X = [yo, hablas, frances] ;
X = [tu, [no_, hablo], aleman] ;
X = [tu, [no_, hablo], frances] ;
X = [tu, hablas, aleman] ;
X = [tu, hablas, frances] ;
```

Práctica: ordenación alfabética

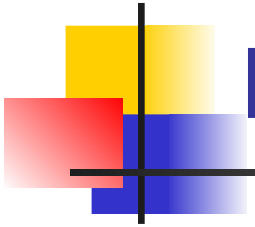
- Las palabras se considerarán una lista de números (enteros) correspondiente a su secuencia ASCII.

- Obtener la lista asociada a cada palabra:

```
?- name(pala, X).
X=[112, 97, 108, 97].
```

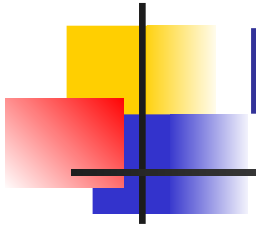
- Comparación:

```
amenor(X, Y) :- name(X, L), name(Y, M), amenorx(L, M).
amenorx([], [_|_]).
amenorx([X|_], [Y|_]) :- X<Y.
amenorx([A|X], [B|Y]) :- A=B, amenorx(X, Y).
```



Práctica: ordenación alfabética

- Ejercicios:
  - ¿qué sucede si se elimina la relación amenor([], [_|_])?
  - Modifique el fichero PROLOG para admitir dentro de “amenor” el caso de dos palabras iguales.



Predicado “append”

- Predicado predefinido:

```
?- append([a, b, c], [1, 2, 3], X).
X=[a, b, c, 1, 2, 3]
?- append(X, [b, c, d], [a, b, c, d]).
X=[a]
?- append([a], [1, 2, 3], [a, 1, 2, 3]).
True
? append([a], [1, 2, 3], [alfa, beta, gamma]).
False
```

- Definición:

```
concatena([], L, L).
concatena([X|L1], L2, [X|L3]) :- concatena(L1,L2,L3)
```