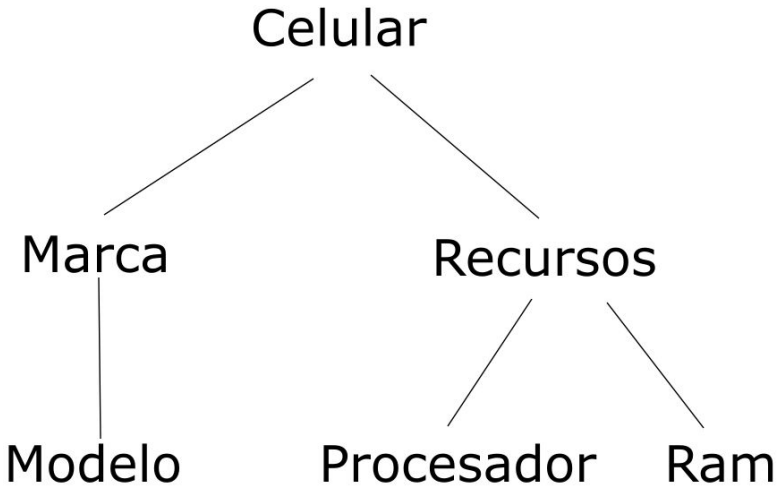
Lógica para Ciencias de la Computación

Laboratorio Prolog: Clase 4

Estructuras de datos: árboles

- La escritura de cualquier estructura de datos como árbol simplifica la interpretación de su forma.
- Utilizando el nombre de la estructura como nodo y sus componentes como ramas obtenemos representaciones simples.



Estructuras de datos: Listas

- Estructura de datos común.
- Es una secuencia ordenada de elementos que puede tener cualquier longitud.
- Los elementos de una lista pueden ser de
  - Cualquier término: Constantes, Variables, Estructuras.
  - Otras listas.

Listas en Prolog

- Pueden representarse como un tipo de especial de árbol.
- Se puede definir recursivamente como:
  - Una lista vacía [], sin elementos.
  - Una estructura de dos componentes:
    - Cabeza: Primer argumento.
    - Cola: Segundo argumento o resto de la lista.
- El final de una lista se puede representar como una cola que contiene la lista vacía. La cabeza y la cola son componentes de una estructura cuyo nombre puede ser definido.

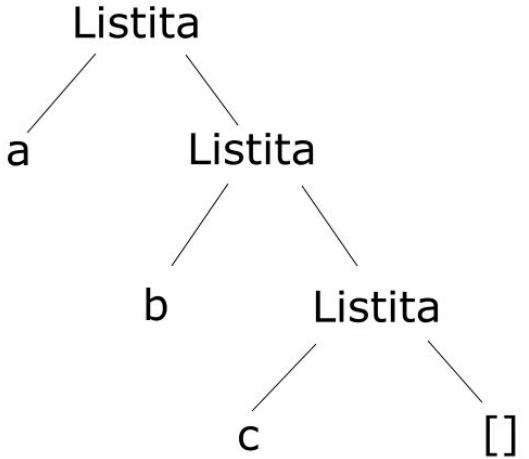
Definición de una lista en prolog

- Definición de una lista que contiene un solo elemento

```
%Lista que contiene un solo elemento
lista(a,[]).
```

- Definición de una lista con varios elementos.

```
%Lista que contiene tres elementos
listita(a,listita(b,listita(c,[]))).
```



Listas en Prolog

- Otra notación más manejable que utiliza Prolog consiste en representar las listas disponiendo a los elementos de la lista separados por comas y toda la lista encerrada entre corchetes.
- De esta manera las listas anteriores quedarían:
  - ***[a]*** y ***[a, b, c]***

Ejemplos de listas

Lista	Cabeza (elemento)	Cola (lista)
[a, b, c]	a	[b, c]
[a]	a	[]
[]	(no tiene)	(no tiene)
[[el perro], ladra]	[el perro]	[ladra]
[el, [perro, ladra]]	el	[perro, ladra]
[el, [perro, ladra], hoy]	el	[[perro, ladra], hoy]
[X+Y, x+y]	X+Y	[x+y]

Diseño de procedimientos de manejo de listas

- Como no se sabe de antemano el tamaño de las listas, se debe utilizar recursividad para recorrerlas.
- ¿Como comprobamos si un elemento pertenece a una lista?
  - Esquema de relación: **miembro(***Elem, Lista***)** <- el término *Elem* pertenece a la lista *Lista*.
  - Definición intuitiva: Un determinado naipe se encuentra en un mazo si es el primero o si está en el resto del mazo.

Llevemoslo a Prolog

- Traducción literal (menos óptimo).

```
%El elemento se encuentra en la lista si es el primero
miembro(Elem, Lista):- Lista=[X|_], X=Elem.

%o si se encuentra en el resto de la lista
miembro(Elem, Lista):- Lista=[_|Y], miembro(Elem,Y).
```

- modificación más óptima

```
%Ahorro de espacio de memoria
miembro(X,[X|_]).
miembro(X,[_|Y]):- miembro(X,Y).
```

Operaciones con listas (i)

- Consulta si un elemento se encuentra en una lista.

```
miembro(X,[X|_]).
miembro(X,[_|Y]):- miembro(X,Y).
```

- Número de elementos de una lista.

```
numElem([],0).
numElem([X|Y],N) :- numElem(Y,M), N is M+1.
```

- Determinar si un elemento es o no una lista.

```
esLista([]).
esLista([_|_]).
```

Operaciones con listas (ii)

- Concatenar dos listas.

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

- Obtener el último elemento de la lista.

```
ultimoElem(X,[X]).
ultimoElem(X,[_|Y]):- ultimo(X,Y).
```

- Obtener la inversa de la lista.

```
listaInversa([],[]).
listaInversa([X|Y],L):- listaInversa(Y,Z), concatLista(Z,[X],L).
```

Operaciones con listas (iii)

- Eliminar un elemento de una lista.

```
borrarElem(X,[X|Y],Y).
borrarElem(X,[Z|L],[Z|M]):- borrarElem(X,L,M).
```

- Consultar si L1 es subconjunto de L2.

```
subconjunto([X|Y],Z):- miembro(X,Z), subconjunto(Y,Z).
subconjunto([],_).
```

- Insertar un elemento en una lista.

```
insertar(Elem, L, [Elem,L]).
insertar(Elem, [X|Y],[X|Z]):- insertar(Elem,Y,Z).
```

Operaciones con listas (iv)

- Comprueba si la lista L2 es una permutación de la lista L1.

```
esPermutacion([],[]).
esPermutacion([X|Y],Z):- esPermutacion(Y,L), insertar(X,L,Z).
```

Algunos experimentos (i)

?- miembro(d,[a,b,c,d,e]).

true ;

false.

?- miembro(k,[a,b,c,d,e]).

false.

?- miembro(d,[a,b,c,[d,e]]).

false.

?- miembro([X,e],[a,b,c,[d,e]]).

X = d ;

false.

?- miembro(X,[a,b,c,d,e]).

X = a ;

X = b ;

X = c ;

X = d ;

X = e ;

false.

Algunos experimentos (ii)

```
?- numElem([a,b,[c,d],e],N).
N = 4.

?- esLista([a,b,[c,d],e]).
```

Ejemplo más práctico (i)

```
?- crear(Lista),apilar(a,Lista,Lista2),apilar(b,Lista2,Lista3),apilar(c,Lista3,Lista4),miembro(a,Lista4,X).
Lista = [],
Lista2 = [a],
Lista3 = [b, a],
Lista4 = [c, b, a],
X = yes ;
false.

?- crear(Lista),apilar(a,Lista,Lista2),apilar(b,Lista2,Lista3),apilar(c,Lista3,Lista4),miembro(z,Lista4,X).
false.
```

Ejemplo más práctico (ii)

?- crear(Lista),apilar(a,Lista,Lista2),apilar(b,Lista2,Lista3),apilar(c,Lista3,Lista4),numElem(Lista4,N).
Lista = [],
Lista2 = [a],
Lista3 = [b, a],
Lista4 = [c, b, a],
N = 3.

?- crear(Lista),apilar(a,Lista,Lista2),apilar(b,Lista2,Lista3),apilar(c,Lista3,Lista4),concatLista(Lista4,Lista3,X).
Lista = [],
Lista2 = [a],
Lista3 = [b, a],
Lista4 = [c, b, a],
X = [c, b, a, b, a].

Actividad

- Compruebe el funcionamiento de las otras operaciones con listas. Utilizar listas diferentes al ejemplo.
- Escribir un procedimiento que obtenga el elemento que ocupa la posición n de una lista o la posición que ocupa el elemento e.
- Escribir un procedimiento que borre el elemento que ocupa la posición n de una lista.

```
?- elemento(E,3,[a,b,c,d]).
E=c

?- elemento(c,N,[a,b,c,d]).
N=3
```

```
?- borrar(3,[a,b,c,d],L).
L=[a,b,d]
```