PLP - Segundo Parcial - 1er cuatrimestre de 2019

Ejercicio 1 - Programación Orientada a Objetos

a. Definir en Cálculo ς una clase *ContadorGente* para modelar un contador de personas que ingresan a un lugar. La clase deberá contar con los siguientes pre-métodos:

- *cantidad*: indica cuántas personas hay adentro (inicialmente en 0).
- *entrar*: aumenta en 1 la cantidad de personas adentro.
- *salir*: disminuye en 1 la cantidad de personas adentro.

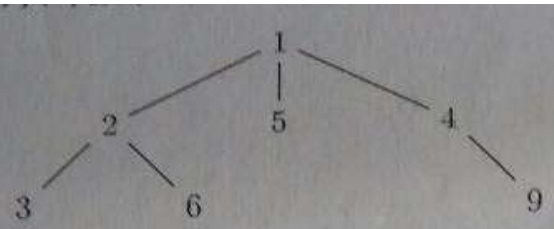
b. Definir un objeto *c* de la clase *ContadorGente*. Mostrar la reducción del término $(c.entrar).cantidad$, indicando en cada paso las reglas utilizadas.

c. Definir, utilizando la clase *ContadorGente*, una clase $ContadorGenteMax$ que agregue el pre-método *maximo*, el cual indica el máximo número de personas que alguna vez hubo adentro (inicialmente en 0). Notar que también se debe modificar *entrar* para poder actualizar el máximo.

Ejercicio 2 - Programación Lógica

Implementar los predicados respetando en cada caso la instanciación pedida. Los generadores deben cubrir todas las instancias válidas de aquello que generan sin repetir dos veces la misma. No usar cut (!) ni predicados de alto orden como `setof`, con la única excepción de `not`.

tupla [1,[2,[(3,[]),(6,[])]),(5,[]),(4,[(9,[])])])



a) Definir el predicado `camino(+R,-C)` que es verdadero cuando `C` es una lista que representa un camino del Rosetree `R` desde el nodo raíz hasta una hoja. `C` debe instanciarse con todos los posibles caminos de `R`. Por ejemplo:

```
?- camino((1,[(2,[(3,[]),(6,[])]),(5,[]),(4,[(9,[])])]),C).
C = [1, 2, 3];
C = [1, 2, 6];
C = [1, 5];
C = [1, 4, 9];
false.
```

b) Definir el predicado `caminoDeMayorValor(+R,-C)` que es verdadero cuando `C` es el camino cuyos nodos suman el valor máximo del Rosetree `R`. Por ejemplo:

```
?- caminoDeMayorValor((1,[(2,[(3,[]),(6,[])]),(5,[]),(4,[(9,[])])]),C).
C = [1, 4, 9];
false.
```

c) El predicado `camino3(R,C)` ¿es reversible en C? Justificar.

Ejercicio 3 - Resolución

Dada la siguiente base de conocimientos en Prolog:

```
member(X,[X|_]).
member(X,[_|XS]) :- member(X,XS).

sacar(X,[X|XS],XS).
sacar(X,[Y|YS],[Y|XS]) :- sacar(X,YS,XS).
```

a) Expresar la base de conocimientos como fórmulas lógicas.

b) Demostrar utilizando resolución que el predicado `sacar` no hace lo esperado. Es decir:

$\exists X \; \exists \; XS \; \exists L \; (\texttt{sacar}(X, XS, L) \; \wedge \; \texttt{member}(X, L))$

c) La resolución utilizada en el punto anterior, ¿fue SLD? Justificar.