Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3661 - Taller de Lenguajes de Programación I
Enero-Marzo 2016

Quiz Prolog

(10 puntos)

Nombre:

Carnet:

Atención: en todas sus respuestas debe utilizar *exclusivamente* predicados disponibles en el estándar ISO provisto por GNU Prolog. Si Ud. no sabe si un predicado está o no en el conjunto estándar ISO provisto por GNU Prolog, puede preguntarlo *públicamente* para recibir un **si** o **no** como respuesta y **nada** más.

1. **(3 puntos)** Escriba el predicado `magic/0` que triunfa después de imprimir *todos* los Cuadrados Mágicos de 3 x 3.

```
?- magic.
 2 | 7 | 6
---+---+---
 9 | 5 | 1
---+---+---
 4 | 3 | 8
...
yes
```

La solución más simple puede ser

```
magic(Square) :-
        % Generate
         Digits = [1,2,3,4,5,6,7,8,9],
         Square = [A,B,C,D,E,F,G,H,I],
         assign_digits(Square,Digits),
        % Test
         T is A + B + C,
         T is D + E + F,
         T is G + H + I,
         T is A + D + G,
         T is B + E + H,
         T is C + F + I,
         T is A + E + I,
         T is C + E + G,
        % Print
         format(' ~d | ~d | ~d',[A,B,C]), nl,
         format('---+---+---',[]), nl,
         format(' ~d | ~d | ~d',[D,E,F]), nl,
         format('---+---+---',[]), nl,
         format(' ~d | ~d | ~d',[G,H,I]), nl.
magic :- magic(_), nl, fail. magic.
```

2. **(2 puntos)** Considere una lista no vacía en la cual aparecen los átomos `r` (por «rojo»), `a` (por «azul») y `b` (por «blanco»), en cualquier orden y repetidos una cantidad arbitraria de veces. El problema de la «Bandera Holandesa» consiste en ordenar todos los átomos de la lista de manera que aparezcan primero todos los `r`, luego todos los `b` y luego todos los `a`, pues ese es el orden de los colores en la bandera. Implante el predicado `flag(Scrambled,Ordered)` que triunfe *exactamente una vez*, si `Ordered` corresponde a organizar los átomos de los elementos de `Scrambled` para formar la «Bandera Holandesa». El argumento `Scrambled` *siempre* debe estar unificado con una lista. Por ejemplo,

```
?- flag([r,b,a,r,b],B).
B = [r,r,b,b,a]
yes
?- flag([],B).
no
?- flag([r,a,r,a,r],[r,r,r,a,a]).
yes
```

Exactamente un recorrido sobre la lista, repartiendo los elementos en tres acumuladores, para luego terminar conectando los tres en el orden necesario. Nótese el uso del cut una vez encontrada la solución, para evitar *backtracking*.

```
flag(Scrambled,Sorted) :- nonvar(Scrambled),
                          flag(Scrambled,Sorted,[],[],[]), !.
flag([],[S|Ss],Rs,Ws,Bs)    :- append(Rs,Ws,X), append(X,Bs,[S|Ss]).
flag([r|R],Sorted,Rs,Ws,Bs) :- flag(R,Sorted,[r|Rs],Ws,Bs).
flag([b|R],Sorted,Rs,Ws,Bs) :- flag(R,Sorted,Rs,[b|Ws],Bs).
flag([a|R],Sorted,Rs,Ws,Bs) :- flag(R,Sorted,Rs,Ws,[a|Bs]).
```

3. **(5 puntos)** Implante el predicado `before(This,That,List)` que triunfe si `This` aparece antes que `That` en la lista `List`. El predicado debe fallar si `This` no aparece en la lista. Además, el predicado debe operar con *cualquiera* de sus dos primeros argumentos libres. Por ejemplo,

```
?- before(1,3,[1,2,3]).
yes
?- before(1,2,[1,3,4,1]).
no
?- before(3,2,[1,2,4,1]).
no
?- before(1,2,[1,3,1,2]).
yes% dos soluciones
?- before(A,B,[1,2,3]).
A = 1
B = 2% tres soluciones
```

Pista: Asegure la presencia y posición de `This`, pero sin usar recursión.

No hace falta contar, no hace falta usar la longitud de la lista, no hace falta averiguar posiciones numéricas, y tampoco hace falta usar recursión. Con `append/3` se fijan *todas* las posiciones de `This` en la lista, y luego se usa `member/2` para verificar la presencia de `That` en los respectivos sufijos.

```
before(This,That,L) :- append(P,[This|R],L), member(That,R), This \= That.
```