Examen resuelto de Programación declarativa (7–02–2002):

Primera parte: Examen escrito.

Ejercicio 1: Se considera el siguiente programa:

```
p([],0).
p([X|L],s(N)) :- p(L,N).
```

Se pide:

1. Escribir el árbol de resolución para la pregunta `?- p([a,b],M).`

```
p([a,b],M)
    |
    | {M/s(N1), X1/a, L1/[b]}
    |
p([b],N1)
    |
    | {N1/s(N2), X2/b, L2/[]}
    |
p([],N2)
    |
    | {N2/0}
    |
   Yes
M = s(s(0))
```

2. Escribir el árbol de resolución para la pregunta `?- p(L,s(s(0))).`

```
p(L,s(s(0)))
    |
    | {L/[X1|L1], N1/s(0)}
    |
p(L1,s(0))
    |
    | {L1/[X2|L2], N2/0}
    |
p(L2,0)
    |
    | {L2/[]}
    |
   Yes
L = [X1,X2]
```

Ejercicio 2: Define el predicado `elimina_vocales(P1,P2)` que tome como dato de entrada una palabra `P1` y devuelva la palabra que se obtiene al eliminar todas las vocales de `P1`.

```
elimina_vocales(P1,P2) :-
   name(P1,L1),
   codigos_vocales(L),
   findall(N,(member(N,L1),not(member(N,L))),L2),
   name(P2,L2).
codigos_vocales(L) :-
   name(aeiou,L).
```

Ejercicio 3: Define el predicado `max_lista(L1,Max)` que recibe como entrada una lista y devuelve o bien el mayor número que es miembro de la lista, o bien `0`, si ningún número es miembro de la lista.

(•) Primera solución:

```
max_lista_1([],0).
max_lista_1([N|R],M):-
   number(N),
   !,
   max_lista_1(R,M1),
   M is max(N,M1).
max_lista_1([_|R],M):-
   max_lista_1(R,M).
```

(•) Segunda solución:

```
max_lista_2(L,M) :-
   member(M,[0|L]),
   number(M),
   not((member(N,L),
        number(N),
        N>M)).
```

Segunda parte: Laboratorio.

Ejercicio L1: Define el predicado `es_cubo(N)` que reciba como dato de entrada un entero positivo `N` y se verifique si `N` es el cubo de un entero.

(•) Primera solución: Muy ineficiente.

```
es_cubo1(N) :-
    between(1,N,X),
    N is X*X*X.
```

```
| ?- time(es_cubo1(1707)).
| 1,709 inferences in 0.06 seconds
| No
```

(•) Segunda solución: Como la anterior, pero acotamos la búsqueda.

```
es_cubo2(N) :-                    | ?- time(es_cubo2(1707)).
    Cota is round(N^(1/3)),       | 15 inferences in 0.00 seconds
    between(1,Cota,X),            | No
    N is X*X*X.                   |
```

(•) Tercera solución: usar predicados aritméticos predefinidos.

```
es_cubo3(N) :-                    | ?- time(es_cubo3(1707)).
    X is integer(N^(1/3)),        | 3 inferences in 0.00 seconds
    N is X*X*X.                   | No
```

(•) Cuarta solución: Generamos la sucesión de los cubos (1,8,27,64,...). `N` es un cubo si el primer cubo mayor o igual que `N` es el propio `N`.

```
es_cubo4(N) :-                    | ?- time(es_cubo4(1707)).
    es_cubo4_aux(N,1).            | 48 inferences in 0.00 seconds
es_cubo4_aux(N,M) :-              | No
    N is M*M*M.                   |
es_cubo4_aux(N,M) :-              |
    M*M*M < N,                    |
    Suc is M+1,                   |
    es_cubo4_aux(N,Suc).          |
```

Ejercicio L2: Define el predicado `descompone(N,Cubo_1,Cubo_2)` que toma como entrada un entero positivo `N` y devuelve los números `Cubo_1` y `Cubo_2`, que son cubos, su suma es `N` y, además, `Cubo_1` es menor o igual que `Cubo_2`.

(•) Primera solución: Muy ineficiente.

```
descompone1(N,Cubo_1,Cubo_2) :-   | ?- time(descompone1(1707,_,_)).
    between(1,N,Cubo_1),          | 35,126,461 inferences in 58.82 seconds
    between(1,N,Cubo_2),          | No
    es_cubo2(Cubo_1),             |
    es_cubo2(Cubo_2),             |
    Cubo_1 =< Cubo_2,             |
    N is Cubo_1+Cubo_2.           |
```

(•) Segunda solución.

```
descompone2(N,Cubo_1,Cubo_2) :-   | ?- time(descompone2(1707,_,_)).
    between(1,N,Cubo_1),          | 20,601 inferences in 0.06 seconds
    es_cubo2(Cubo_1),             | No
    Cubo_2 is N - Cubo_1,         |
    Cubo_1 =< Cubo_2,             |
    es_cubo2(Cubo_2).             |
```

(•) Tercera solución: Como la anterior, pero acotamos la búsqueda.

```
descompone3(N,Cubo_1,Cubo_2) :-
    Cota is round((N/2)^(1/3)),
    between(1,Cota,M),
    Cubo_1 is M*M*M,
    Cubo_2 is N-Cubo_1,
    Cubo_1 =< Cubo_2,
    es_cubo2(Cubo_2).
```

```
| ?- time(descompone3(1707,_,_)).
| 160 inferences in 0.00 seconds
| No
|
|
|
|
|
```

Ejercicio L3: Define el predicado `ramanujan(N)` que toma como entrada un entero positivo `N` y se verifica si `N` puede descomponerse en suma de dos cubos *exactamente* de dos maneras distintas.

```
ramanujan(N) :-
    setof(par(Cubo_1,Cubo_2),descompone3(N,Cubo_1,Cubo_2),[_,_]).
```

Ejercicio L4: Define el predicado `hardy(N)` que devuelve el menor entero positivo que satisface el predicado `ramanujan` anterior.

```
hardy(N) :-
    hardy_aux(N,1).
hardy_aux(N,N) :-
    ramanujan(N),
    !.
hardy_aux(N,M) :-
    Suc is M+1,
    hardy_aux(N,Suc).
```

```
| ?- time(hardy(N)).
| 207,742 inferences in 0.33 seconds
| N = 1729
|
|
|
|
|
```