UNIVERSIDAD DE MÁLAGA
Departamento de Lenguajes y Ciencias de la Computación

Examen de Programación Declarativa
Junio de 2001
Ejercicio de Programación Funcional

Alumno:_____ **Grupo**:_____ **Fecha**:_____

Ejercicio 1: (*0,75 + 1,75 ptos*)

A) Enunciad formalmente el principio de inducción estructural aplicado a la siguiente propiedad:

```
                concat (xs++ys) = concat xs ++ concat ys
    xs :: [[a]].  ys ::[[a]] . concat (xs++ys) = concat xs ++ concat ys
    =
    ys ::[[a]]. concat ([]++ys) = concat [] ++ concat ys
    ^
    x::[a+,xs::[[a]]. ys::[[a]]. concat (xs++ys) = concat xs ++ concat ys
          =>
          ys::[[a]]. concat (x:xs++ys) = concat (x:xs) ++ concat ys
```

B) Demostrad la propiedad anterior sabiendo que el operador ++ es asociativo y que

```
    concat [] = []
    concat (x:xs) = x ++ concat xs
```

```
-- Caso base

      concat ([]++ys)                    concat [] ++ concat ys
= ! 1ª de (++)                     =! 1ª concat
      concat ys                          [] ++ concat ys
                                   = ! 1ª de (++)
                                         concat ys

-- Paso inductivo

      concat (x:xs++ys)                  concat (x:xs) ++ concat ys
= ! 2ª de (++)                     = ! 2 de conccat
      concat (x :(xs++ys))               (x++concat xs) ++ concat ys
= !1ª de concat                    = ! asociatividad de (++)
      x++concat (xs++ys)                 x++(concat xs ++ concat ys)
= ! HI
      x++(concat xs ++ concat ys)
```

Ejercicio 2: (*1,25 + 1,25 ptos*)

Encontrad, usando listas por comprensión, expresiones equivalentes a cada una de las siguientes

```
A) map (+3) (filter even xs)

            [x+3| x <- xs, even x]

B) filter even (map (+3) xs)

            [y | x <- xs,let y = x+3, even y]
```

Ejercicio 3: (*1,25 + 1,25 ptos*)

Dadas las siguientes definiciones de tipo para representar expresiones aritméticas:

```
data Op = Mas | Menos | Por
data Exp = N Int | E Op Exp Exp
```

y la correspondiente función de plegado

```
foldExp f h (N x) = h x
foldExp f h (E o i d) = f o (foldExp f h i) (foldExp f h d)
```

expresad, como casos particulares de `foldExp`

A) Una función `enteros` que calcule el número de enteros de un árbol de tipo Exp.
Ejemplo: `enteros (E Mas (N 3) (N 5))` debe dar `2`

```
enteros::Exp -> Integer
enteros = foldExp (\o x y -> x+y) (\x -> 1)
```

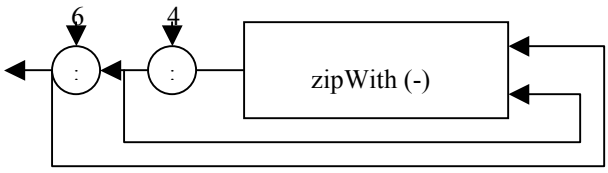
B) Una función `valor` que calcule el valor de un árbol de tipo Exp.
Ejemplo: `valor (E Mas (N 3) (N 5))` debe dar `8`

```
valor::Exp -> Int
valor = foldExp f id
        where  f Mas x y = x+y
               f Menos x y = x-y
               f Por x y = x*y
```

Ejercicio 4: (*1,25 + 1,25 ptos*)

A) Construid una red de procesos, junto con su correspondiente expresión Haskell, para generar la siguiente lista infinita `as = [6,4,2,2,0,2,-2,4,-6,...]`
en la que cada elemento se obtiene a partir de los dos anteriores restándolos.

```
as = 6:4:zipWith (-) as (tail as)
```



B) Definid una función recursiva `emparejar` para emparejar los elementos de una lista de forma alternada.
Ejemplo: `emparejar as` debe dar `[(6,2),(4,2),(0,-2),(2,4),...]`

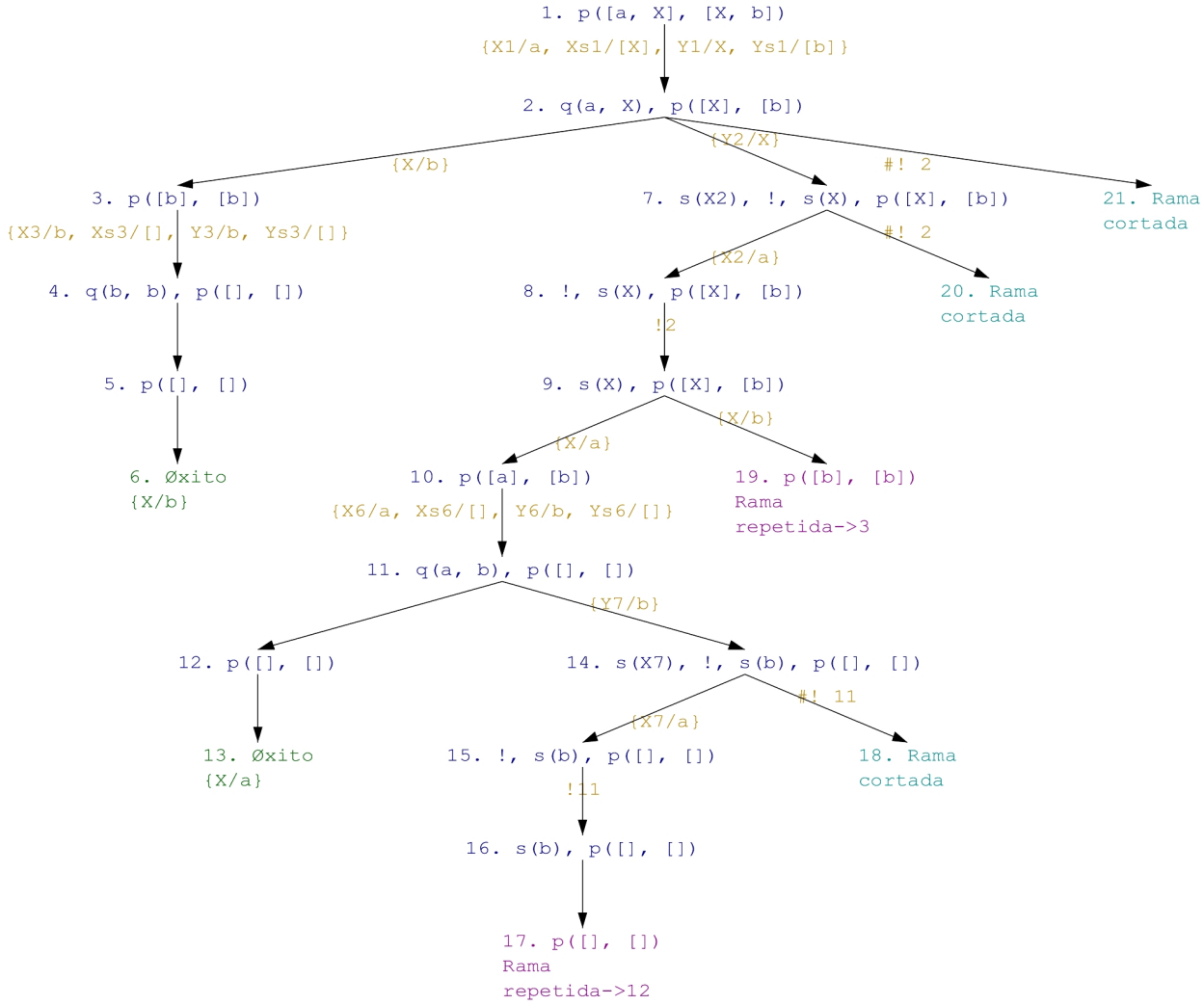
```
emparejar (x:y:z:t:xs) = (x,z):(y,t):emparejar xs
```

PROLOG

Ejercicio 1. (*2 + 0,5 ptos*) Dado el programa:

```
p([],[]).
p([X|Xs],[Y|Ys]):- q(X,Y), p(Xs,Ys).
q(a,b).
q(a,Y) :- s(X), !, s(Y).
q(a,c).
q(b,b).
s(a).
s(b).
```

a) representa el árbol de búsqueda **SLD** correspondiente al objetivo **`p([a,X],[X,b]).`**



b) ¿Qué soluciones proporcionaría Prolog en este caso y en qué orden?

{X/b}, y después {X/a} indefinidamente

Ejercicio 2. (*1,25+1,25 ptos*) El predicado **`purga(Xs,Ys)`** se satisface cuando **`Ys`** es la lista formada por los elementos de la lista **`Xs`**, **sin repeticiones**. Por ejemplo, se satisface el objetivo:

`purga( [1,2,3,1,2,3,4,4,5],[1,2,3,4,5])`

a) escribe una versión de **`purga/2`** recursiva no de cola

```
purga([],[]).
purga([X|Xs],Ys) :-
      purga(Xs,Ys),
      member(X,Ys).
purga([X|Xs],[X|Ys]) :-
      purga(Xs,Ys),
      \+ member(X,Ys).
```

b) escribe una versión de **purga/2** iterativa (es decir, con recursión de cola)

```
purga_cola(Xs,Ys):- purga_aux(Xs,[],Ys).
purga_aux([],Ac,Ac).
purga_aux([X|Xs],Ac,Y) :-
      member(X,Ac),
      !,
      purga_aux(Xs,Ac,Y).
purga_aux([X|Xs],Ac,Y) :-
      append(Ac,[X],NAc),
      purga_aux(Xs,NAc,Y).
```

Ejercicio 3. (*2,5 ptos*) Describir la tabla de comportamiento del predicado **p/3**

```
p(X,Xs,Ls):-
      append(As,[X|Bs],Xs),
      append(Bs,As,Ls).
```

Modos de Uso	Comportamiento	Significado
(+,+,+)	test	Comprueba que la lista Ls es obtenida partiendo la lista Xs en dos partes, quitando de la cabeza de la segunda el elemento X y juntándolas, primero la segunda y luego la primera
(+,+,-)	Generador acotado	Crea listas Ls que resultan de partir en dos Xs, quitar el elemento X de la cabeza de la segunda y juntándolas, primero la segunda y luego la primera
(-,+,-)	Generador acotado	Crea listas Ls que resultan de partir la lista Xs en dos, extraer la cabeza de la segunda (que se instancia al primer argumento) y juntando las listas, primero la segunda y luego la primera
(+,-,+)	Generador anómalo	Hace un uso anómalo del predicado append

Ejercicio 4. (*2,5 ptos*) Escribe un predicado **funtores(T,F)** que se satisfaga cuando **F** sea la lista **sin repeticiones** de los funtores (con aridad >0), representados como funtor/aridad que aparecen en el término **T**. Por ejemplo:

```
?- funtores(f(g(a),f(X,Y),t(g(Y),5)),F).
F= [f/3,g/1,f/2,t/2];
No
```

```
funtores(T,F) :-
      funtores_rep(T,FRep),
      purga(FRep,F).
funtores_rep(A,[]) :-
      atomic(A).
funtores_rep(V,[]) :-
      var(V).
funtores_rep(S,[F/N|Fs]) :-
      compound(S),
      S =.. [F|Args],
      length(Args,N),
      funtores_rep_args(Args,Fs).
funtores_rep_args([],[]).
funtores_rep_args([A|As],FArgs) :-
      funtores_rep(A,Fa),
      funtores_rep_args(As,Fas),
      append(Fa,Fas,FArgs).
```