UNIVERSIDAD DE MÁLAGA
E.T.S.I. INFORMÁTICA

Programación Declarativa

(3º de Ingeniería Técnica en Informática)
Julio de 2005

Alumno: _____ Grupo: _____

Prolog

Ejercicio 1

$(a)(2\ pts.)$ Realiza el árbol de búsqueda para el objetivo `p([a,b,c,b,a],X)` y el programa *Prolog*

```
p([X],X).
p([X|Xs],Y) :-
        concatena(As,[X],Xs),
        p(As,Y).
concatena([],Ys,Ys).
concatena([X|Xs],Ys,[X|Zs]):-
        concatena(Xs,Ys,Zs).
```

$(b)(1\ pto.)$ Completa la tabla de comportamiento del predicado `p` anterior.

$(+,+)$		
$(+,-)$		
$(-,+)$		
$(-,-)$		

Ejercicio 2

$(a)(1{,}25\ pts.)$ Define el predicado `genNat/1` que genera los números naturales:

```
?- genNat(N).
N = 0;
N = 1;
N = 2;
...
```

$(b)(1{,}25\ pts.)$ Define el predicado `genEnt/1` que genera los números enteros (naturales con signo):

```
?- genEnt(N).
N = 0;
N = 1;
N = -1;
N = 2;
...
```

Ejercicio 3

$(1{,}5\ pts.)$ Define el predicado `posiciones(AsXsBs,Xs,N)` tal que devuelva por revaluación en `N` las posiciones en las que aparece la sublista `Xs` en la lista `AsXsBs`.

```
?- posiciones([a,b,a,b,a,b,c,a,b,a],[a,b,a],N).
N  = 0;
N  = 2;
N  = 7;
No
```

Ejercicio 4

$(a)(1{,}5\ pts.)$ Define el predicado `cuenta(XsYs,X,Ys,N)` tal que dadas la lista `XsYs` y un término `X`, devuelva en `N` el número de `X` que aparecen al principio de `XsYs`, y en `Ys` la lista que resulta de eliminar todas las `X` por las que empieza `XsYs`

```
?- cuenta([5,5,5,4,4,1,6],5,N,Ys).
N = [4,4,1,6],
Ys = 3  ;
No
?- cuenta([5,5,5,4,4,1,6],1,N,Ys).
N = [5,5,5,4,4,1,6],
Ys = 0  ;
No
```

$(b)(1{,}5\ pts.)$ Define el predicado `nombra(Xs,Ys)` que dada una lista de enteros positivos `Xs`, devuelve una lista `Ys` que "nombra" la lista `Xs`, mencionando cuantas veces consecutivas aparece cada elemento de `Xs`

```
?- nombra([1,1,1,3,3,7,4,4,3],Ys).
Ys = [3,1,2,3,1,7,2,4,1,3] ;
No
```

UNIVERSIDAD DE MÁLAGA
E.T.S.I. INFORMÁTICA

Programación Declarativa

(3º de Ingeniería Técnica en Informática)
Julio de 2005

Alumno: _____ Grupo: _____

Haskell

Ejercicio 1

$(a)(0{,}75\ pts.)$ ¿ Cuál es el tipo polimórfico de la función `h` ?

```
h f f' g (x,y) = g (f x, f' y)
```

```
h :: _____
```

$(b)(0{,}75\ pts.)$ ¿Cuál es el tipo polimórfico de la función `h'` ?

```
h' f g (x,y) = (g (f x), f (g y))
```

```
h' :: _____
```

$(c)(0{,}5\ pts.)$ ¿Cuál es el tipo polimórfico de la función `f` ?

```
f g p h = filter p . map g . map h
```

```
f :: _____
```

$(d)(0{,}75\ pts.)$ Define la función `f` del apartado anterior usando una lista por comprensión

```
f g p h = _____
```

$(e)(0{,}75\ pts.)$ Define la función `f` del apartado anterior usando `foldr`

```
f g p h = foldr_____
            where
                _____
                _____
                _____
                _____
```

Ejercicio 2

$(a)(1{,}5\ pts)$ Define una función `palabra :: (Char->Bool) -> String -> String` que extraiga la primera palabra de una lista de caracteres. La palabra estará formada por el mayor segmento inicial de caracteres que no sean separadores. El primer parámetro de la función `palabra` es una función que dado un caracter devuelve `True` si éste es un separador. Por ejemplo

```
palabra (==' ') "Esto es un texto" ==> "Esto"
palabra (\c -> c `elem` " ,.") "Esto, es un texto" ==> "Esto"
```

$(b)(1{,}5\ pts)$ Define una función `palabras :: (Char->Bool) -> String -> [String]` que extraiga todas las palabras de una cadena de caracteres. Por ejemplo

```
palabras (==' ') "Esto es un texto"  ==> ["Esto","es","un","texto"]
palabras (\c -> c `elem` " ,.") "Esto, es un texto."
  ==> ["Esto","es","un","texto"]
```

Ejercicio 3

Sea la definición de árboles no vacíos con información en nodos y hojas siguiente

```
data Arbol a = Hoja a | Nodo (Arbol a) a (Arbol a) deriving Show
```

$(a)(1\ pt)$ Define de modo recursivo la función `sumNH` que devuelva un par cuya primera componente es la suma de los valores de los nodos y su segunda componente es la suma de los valores de las hojas. Da el tipo mas general de la función `sumNH`.

$(b)(2\ pts)$ Redefine la función anterior utilizando el siguiente plegado

```
plegadoArbol :: (b -> a -> b -> b) -> (a -> b) -> Arbol a -> b
plegadoArbol f g (Hoja x)     = g x
plegadoArbol f g (Nodo i x d) = f (plegadoArbol f g i) x (plegadoArbol f g d)
```

que también puede escribirse como

```
plegadoArbol f g = fun
    where fun (Hoja x)     = g x
          fun (Nodo i x d) = f (fun i) x (fun d)
```