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

Programación Declarativa

(3º de Ingeniería Técnica en Informática)
Febrero de 2007

Nombre:
Especialidad: **Grupo:**

Prolog

Ejercicio 1.1 *(1.5 puntos)*

Realizad el árbol de búsqueda con el objetivo `p(b,X,Y).` y el siguiente programa *Prolog*.

```
p(X,X,Y):-p(Y,X,a).
p(X,a,b):-q(X,a).
q(Y,Y):-t(Y).
q(Y,X):-q(a,X).
t(b).
t(a).
```

Ejercicio 1.2 *(1 punto)*

¿Cuál/es será/n la/s respuesta/s de *Prolog* para el caso anterior?.

Ejercicio 2.1 *(1 punto)*

Dad una definición del predicado `mayor/2` que sea cierto cuando dada una lista primer argumento `Xs` y un elemento `X` de dicha lista, ese elemento resulte ser mayor que todos sus predecesores en la lista. (Define todos los predicados que necesites) Por ejemplo

```
?- mayor([5,4,9,2,67,4,3,2,1],X).
    X = 5 ;
    X = 9 ;
    X = 67 ;
    No
```

Ejercicio 2.2 *(1,5 puntos)*

Define el predicado `listam/2` que devuelva en su segundo argumento una lista con los elementos de la lista primer argumento que verificaban el predicado anterior. Ejemplo:

```
?- listam([5,4,9,2,67,4,3,2,1],Xs).
    Xs = [5, 9, 67];
    No
```

Ejercicio 3 *(2,5 puntos)*

Realizad el predicado `entre(X,Y,Z,Xs)` que se satisfaga cuando `X` se encuentre entre `Y` y `Z` en la lista `Ls`. Define cualquier predicado utilizado.

Ejercicio 4 *(2,5 puntos)*

Disponemos de una lista `Ops` de operadores binarios aritméticos, una lista `Ns` de números y un número `N` objetivo. Queremos diseñar un predicado `reduce(Ops,Ns,N)` que implemente un algoritmo recursivo para reducir las listas de operadores y números hasta que la primera quede vacía y la segunda tenga un sólo elemento que debe coincidir con el número buscado `N`. En cada paso de reducción, se aplica un operador de la lista de operadores a dos operandos de la lista de números y se almacena el resultado en esta lista de números. Por ejemplo:

```
   Ops            Ns               N
[+,-,+,*]    [5,3,6,5,2]         -20
    -              3 5
 [+,+,*]      [-2,5,6,2]          -20
    +              5 6
  [+,*]        [11,-2,2]          -20
    *             11 -2
   [+]          [-22,2]           -20
    +            -22 2
   []             [-20]           -20
```

Define todos los predicados que utilices.

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

Programación Declarativa

(3º de Ingeniería Técnica en Informática)
Febrero de 2007

Nombre:
Especialidad: **Grupo:**

Haskell

Ejercicio 1

Considera la siguiente función:

```
fun f xs = [ y | x <- xs, let y = f x, f y ]
```

- *(1 punto)* ¿Cuál es su tipo?

- *(1 punto)* Da una definición alternativa usando `map`, `filter` y `(.)`.

Ejercicio 2

Dada la siguiente definición de tipo: `type Cromosoma = [Bool]`

- *(1 punto)* Expresa en Haskell la lista infinita de potencias de 2: `[1,2,4,8,...]`

- *(1 punto)* Define una función
  `filterCrom :: Cromosoma -> [Integer]`
  que, a partir de un cromosoma filtre la lista de potencias de 2 manteniendo las potencias que coinciden con posiciones a `True` en el cromosoma y cambiando por `0` las que coinciden con posiciones a `False`, como en el ejemplo siguiente:
  `filterCrom [True,False,False,True,True,False] ==> [1,0,0,8,16,0]`

- *(1 punto)* Define usando `foldr` una función
  `fitness :: Cromosoma -> Integer`
  que filtre con el cromosoma de su argumento la lista de potencias de 2 y sume la lista resultante, como en el ejemplo siguiente:
  `fitness [True,False,False,True,True,False] ==> 25`

- *(1 punto)* Define una función
  `cruce :: Cromosoma -> Cromosoma -> Int -> (Cromosoma,Cromosoma)`
  para cruzar el contenido de dos cromosomas a partir de una posición dada. Así `cruce c1 c2 n`, cuando `n` es una posición válida en el cromosoma más corto, producirá un par de cromosomas: el primero formado con las primeras posiciones de `c1`, hasta la posición `n` inclusive, y el resto de posiciones de `c2` desde la posición `n+1`, y el segundo formado con las primeras posiciones de `c2`, hasta la posición `n` inclusive, y el resto de posiciones de `c1` desde la posición `n+1`. Cuando `n` no es una posición válida en alguno de los cromosomas, éstos no se cruzan y quedan en el mismo orden en que aparecen en los argumentos.
  (**Nota:** Se valorará una definición recursiva)

  Por ejemplo:

```
cruce [True, False, False, True, True]
      [True, True, False, False, False, True] 4
=> ( [True, False, False, True, False, True]
   , [True, True, False, False, True]
   )
```

Ejercicio 3

Sean los tipos

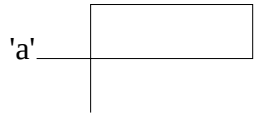
```
data Mov = N Int | S Int | E Int | O Int

data Paseo a = Pasos Mov (Paseo a)| Descansa a
```

El primero define un movimiento en una dirección cardinal, norte, sur, este y oeste. Ese movimiento contiene un número de pasos en la dirección dada (el entero indica el número de pasos a dar).

El segundo representa un paseo que realiza un dato de tipo `a` como una secuencia de movimientos recordando todos los pasos y cambios de direcciones. Por ejemplo:

```
Pasos (N 20) (Pasos (E 30) (Pasos (S 10) (Pasos (O 40) (Descansa 'a'))))
```

representa un paseo que avanza 20 pasos al norte, 30 al este, 10 al sur, 40 al oeste y finalmente descansa. En este caso, el dato que pasea es la letra `'a'`. La siguiente figura muestra gráficamente el paseo anterior:



- *(1 punto)* Define la función `describe :: [Mov] -> a -> Paseo a` que a partir de una lista de movimientos y un dato, genera un paseo para ese dato con los movimientos que aparecen en la lista y manteniendo el orden. Es decir, el primer elemento de la lista será el primer paseo y así sucesivamente.

- *(1 punto)* Define una función `pos :: Paseo a -> (Int,Int) -> (Int,Int)` en la que dado un paseo y una posición inicial (de coordenadas enteras) en el plano , devuelva la posición final (de coordenadas enteras) en el plano donde descansará el dato tras efectuar el paseo.

- *(1 punto)* Define una función de plegado

  ```
  foldrt :: (Mov -> b -> b) -> (a -> b) -> (Paseo a) -> b
  ```

  para la estructura `Paseo a`

- *(1 punto)* Define la función `pos` como un caso particular del plegado anterior.