Examen de Prácticas - 5 de febrero de 2021
LTP - 2º Parcial (Haskell y Prolog) - Tipo A

ALUMNO: _____ GRUPO: _____

Instrucciones

- El alumno dispone de 60 minutos para resolver el examen.
- El examen consta de 5 preguntas que deberán responderse en el mismo enunciado, en los recuadros incluidos en cada pregunta.

Pregunta 1 – Haskell (2.20 puntos)

Define una función `zipfilter` cuyo tipo es:

```
zipfilter :: Int -> [a] -> [Int] -> [(a,Int)]
```

Dados un entero, `n`, y dos listas de la misma longitud, la primera, `la`, con elementos de tipo `a`, y la segunda, `li`, de enteros, `(zipfilter n la li)` devuelve una lista cuyos elementos son pares de tipo `(a,Int)`, resultado de emparejar, solo para elementos de `li` que sean mayores que `n`, los elementos de `la` y `li` que están en la misma posición.

REQUISITO: Se debe resolver mediante recursión o mediante listas intensionales.

Ejemplo de uso.

```
*Main> let lista1 = ['‘a', 'x', 'z', 'e', 'r', 'q', 'p', 'e', 'z', 'b', 's']
*Main> let lista2 = [  3,   7,   2,   1,   3,   8,   4,   5,   7,   9,   0 ]
*Main> zipfilter 3 lista1 lista2
[('x',7),('q',8),('p',4),('e',5),('z',7),('b',9)]
```

Solución con listas intensionales:

```
zipfilter :: Int -> [a] -> [Int] -> [(a,Int)]
zipfilter n x y = [(x !! i, y !! i) | i <- [0..length x - 1], (y !! i) > n]
```

Solución recursiva:

```
zipfilter :: Int -> [a] -> [Int] -> [(a,Int)]
zipfilter _ [] [] = []
zipfilter n (x:xs) (y:ys)
   | y > n = (x,y) : z
   | otherwise = z
   where z = zipfilter n xs ys
```

Pregunta 2 – Haskell (2.20 puntos)

Considera disponible la siguiente definición:

```
data Tree a = Leaf a | Branch (Tree a) (Tree a) deriving Show
```

Define una función `modify` cuyo tipo es: `modify :: (a -> b) -> Tree a -> Tree b`

Dados una función, `f`, de tipo `(a -> b)`, y un árbol, `ta`, de tipo `Tree a`, `(modify f ta)` devuelve un árbol de tipo `Tree b`, con la misma estructura (en cuanto a número y disposición de hojas y ramas) del árbol `ta` y con los valores resultado de aplicar `f` a los valores de `ta`.

Ejemplos de uso. donde se importa `Data.Char` para poder usar las funciones `chr` y `ord`.

```
*Main> import Data.Char
*Main Data.Char> let t1 = Branch (Leaf 81) (Branch (Leaf 82) (Branch (Leaf 83)
                          (Leaf 84)))
*Main Data.Char> modify chr t1
Branch (Leaf 'Q') (Branch (Leaf 'R') (Branch (Leaf 'S') (Leaf 'T')))
*Main Data.Char> let t2 = Branch (Leaf 'a') (Branch (Leaf 'b') (Branch (Leaf 'e')
                          (Leaf 'f')))
*Main Data.Char> modify ord t2
Branch (Leaf 97) (Branch (Leaf 98) (Branch (Leaf 101) (Leaf 102)))
```

Solución:

```
modify :: (a -> b) -> Tree a -> Tree b
modify f (Leaf x) = Leaf (f x)
modify f (Branch izq der) = Branch (modify f izq) (modify f der)
```

Pregunta 3 – Haskell (2.20 puntos)

Considera disponible la siguiente definición:

```
data Queue a = EmptyQueue | Item a (Queue a) deriving Show
```

Define una función `operate` cuyo tipo es:

```
operate :: (a -> a -> a) -> Queue a -> Queue a -> Queue a
```

Dadas una función, `f`, de tipo `(a -> a -> a)`, y dos colas, `p` y `q`, de tipo `Queue a` y de la misma longitud, `(operate f p q)` devuelve una cola del mismo tipo cuyos elementos son resultado de aplicar `f` a los elementos de `p` y `q` que están en las mismas posiciones.

Ejemplos de uso.

```
*Main> let q1 = Item 97 (Item 98 (Item 101 EmptyQueue))
*Main> let q2 = Item 3 (Item 6 (Item 4 EmptyQueue))
*Main> operate (-) q1 q2
Item 94 (Item 92 (Item 97 EmptyQueue))
```

```
*Main> let q4 = Item "ana" (Item "isabel" (Item "lola" EmptyQueue))
*Main> let q5 = Item "-pepe" (Item "-juan" (Item "-javi" EmptyQueue))
*Main> operate (++) q4 q5
Item "ana-pepe" (Item "isabel-juan" (Item "lola-javi" EmptyQueue))
```

Solución:

```
operate :: (a -> a -> a) -> Queue a -> Queue a -> Queue a
operate _ EmptyQueue EmptyQueue = EmptyQueue
operate f (Item p1 q1) (Item p2 q2) = Item (f p1 p2) (operate f q1 q2)
```

Pregunta 4 – Prolog (1.70 puntos)

Dada la siguiente base de conocimiento:

```
month(ene,1,31). % ene es el primer mes (1), y tiene 31 días
month(feb,2,28). % feb es el segundo mes (2), y tiene 28 días (no consideramos bisiestos)
month(mar,3,31). % mar es el tercer mes (3), y tiene 31 días
month(abr,4,30). % abr es ...
month(may,5,31).
month(jun,6,30).
month(jul,7,31).
month(ago,8,31).
month(sep,9,30).
month(oct,10,31).
month(nov,11,30).
month(dic,12,31).
```

Define un predicado `previous` que permita consultar si un mes es anterior a otro mes.

Ejemplos de uso.

```
?- previous(sep,X).
X = oct ;
X = nov ;
X = dic.
?- previous(sap,X).
false.
```

```
?- previous(X,may).
X = ene ;
X = feb ;
X = mar ;
X = abr .
```

Solución:

```
previous(X,Y) :- month(X,M,_), month(Y,N,_), M < N.
```

Pregunta 5 – Prolog (1.70 puntos)

Dada la siguiente base de conocimiento:

```
% prestamos(DB,F), donde:
%    DB es una lista de pares (P,B), siendo P una persona y B un libro
%    F una fecha de préstamo mediante el functor date(D,M,A)
% Cada hecho prestamos(DB,F) contiene la lista de préstamos DB hechos en una fecha F.
prestamos([("Ana", "Ana Karenina"), ("Juan", "La broma"),
           ("Pepe", "El castillo"), ("Alicia", "Lituma en los Andes")], date(29,dic,2020)).
prestamos([("Alicia", "El nombre de la rosa"), ("Juan", "La hija del canibal"),
          ("Pepe", "Odessa"), ("Alicia", "La ciudad de las bestias")], date(7,ene,2021)).
prestamos([("Pepe", "El pirata"), ("Ana", "Jane Eyre"),
           ("Ana", "El unicornio"), ("Juan", "El nombre de la rosa")], date(21,ene,2021)).
```

Define un predicado `obtain` que permita consultar los préstamos de una persona dada, facilitando la siguiente información: título del libro y fecha del préstamo.
Si se considera necesario, pueden usarse predicados predefinidos como, por ejemplo, `member` o `append`.

Ejemplos de uso.

```
?- obtain("Ana",X).
X =  ("Ana Karenina", date(29, dic, 2020)) ;
X =  ("Jane Eyre", date(21, ene, 2021)) ;
X =  ("El unicornio", date(21, ene, 2021)) .
```

```
?- obtain("Pepe",X).
X =  ("El castillo", date(29, dic, 2020)) ;
X =  ("Odessa", date(7, ene, 2021)) ;
X =  ("El pirata", date(21, ene, 2021)) .
```

Solución:

```
obtain(P,X) :- prestamos(DB,F), member((P,B),DB), X = (B,F).
```