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

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 `concatReverse` cuyo tipo es: `concatReverse :: [a] -> [a]`

Dada una lista, `lx`, del tipo `[a]`, `(concatReverse lx)` devuelve una lista, del mismo tipo, resultado de concatenar `lx` y la inversa de `lx`.

REQUISITOS:
- 1) Se debe resolver mediante recursión o mediante listas intensionales.
- 2) No se permite usar la función predefinida `reverse`.

Ejemplo de uso.

```
*Main> let lista1 = [3, 7, 2, 1, 3, 8, 4, 5, 7, 9, 0]
*Main> concatReverse lista1
[3,7,2,1,3,8,4,5,7,9,0,0,9,7,5,4,8,3,1,2,7,3]
```

Solución con listas intensionales:

```
concatReverse :: [a] -> [a]
concatReverse x = x ++ [x !! (z - i) | i <- [0..z]] where z = length x - 1
```

Solución recursiva:

```
concatReverse :: [a] -> [a]
concatReverse [] = []
concatReverse (x:xs) = x : concatReverse xs ++ [x]
```

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 `operate` cuyo tipo es:

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

Dados una función, `f`, de tipo `(a -> a -> a)`, y dos árboles, `tx` y `ty`, de tipo `Tree a` y con la misma estructura (en cuanto a número y disposición de hojas y ramas), `(operate tx f ty)` devuelve un árbol del mismo tipo, y también con la misma estructura, con los valores resultado de aplicar `f` a los valores de `tx` y `ty`.

Ejemplos de uso.

```
*Main> let t1 = Branch (Leaf 9) (Branch (Leaf 2) (Branch (Leaf 8) (Leaf 4)))
*Main> let t2 = Branch (Leaf 3) (Branch (Leaf 7) (Branch (Leaf 5) (Leaf 6)))
*Main> operate t1 (*) t2
Branch (Leaf 27) (Branch (Leaf 14) (Branch (Leaf 40) (Leaf 24)))
*Main> let t3 = Branch (Leaf "ana-") (Branch (Leaf "isa-") (Branch (Leaf "lola-")
                (Leaf "eva-")))
*Main> let t4 = Branch (Leaf "pepe") (Branch (Leaf "jon") (Branch (Leaf "javi")
                (Leaf "edu")))
*Main> operate t3 (++) t4
Branch (Leaf "ana-pepe") (Branch (Leaf "isa-jon") (Branch (Leaf "lola-javi")
(Leaf "eva-edu")))
```

Solución:

```
operate :: Tree a -> (a -> a -> a) -> Tree a -> Tree a
operate (Leaf x) f (Leaf y) = Leaf (f x y)
operate (Branch i1 d1) f (Branch i2 d2) = Branch (operate i1 f i2) (operate d1 f d2)
```

Pregunta 3 – Haskell (2.20 puntos)

Considera disponible la siguiente definición:

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

Define una función `modify` cuyo tipo es:

```
modify :: (a -> b) -> Queue a -> Queue b
```

Dadas una función, `f`, de tipo `(a -> b)`, y una cola, `q`, de tipo `Queue a`, `(modify f q)` devuelve una cola de tipo `Queue b` cuyos elementos son resultado de aplicar `f` a los elementos de `q`.

Ejemplos de uso. donde se importa `Data.Char` para poder usar la función `chr`.

```
*Main> let q1 = Item "ana" (Item "isabel" (Item "lola" EmptyQueue))
*Main> modify length q1
Item 3 (Item 6 (Item 4 EmptyQueue))
*Main> import Data.Char
*Main Data.Char> let q2 = Item 97 (Item 98 (Item 101 EmptyQueue))
*Main Data.Char> modify chr q2
Item 'a' (Item 'b' (Item 'e' EmptyQueue))
```

Solución:

```
modify :: (a -> b) -> Queue a -> Queue b
modify _ EmptyQueue = EmptyQueue
modify f (Item p q) = Item (f p) (modify f q)
```

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 `validator` que permita comprobar si una fecha especificada mediante el predicado `date(D,M,Y)` es correcta, teniendo en cuenta que solamente se consideran correctas fechas desde el 1-ene-1950 hasta el 31-dic-2030, y que se considera que no hay años bisiestos.

Ejemplos de uso.

```
?- validator(date(5,feb,2021)).
true.
?- validator(date(29,feb,2000)).
false.
?- validator(date(30,mac,2021)).
false.
?- validator(date(12,abr,1942)).
false.
?- validator(date(31,sep,1999)).
false.
?- validator(date(-5,jun,2001)).
false.
```

Solución:

```
validator(date(D,M,Y)) :- Y >= 1950, Y =< 2030, month(M,_,N), D > 0, D =< 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"), ("Alicia", "Lituma en los Andes")], date(29,dic,2020)).
prestamos([("Juan", "La broma"), ("Pepe", "El castillo")], date(32,dic,2020)).
prestamos([("Pepe", "Odessa"), ("Alicia", "La ciudad de las bestias")], date(7,ene,2021)).
prestamos([("Alicia", "El nombre de la rosa"), ("Juan", "La hija del canibal")], date(7,efh,2021)).
prestamos([("Pepe", "El pirata"), ("Ana", "El unicornio")], date(21,ene,2021)).
```

Define un predicado `incorrect` que permita consultar los préstamos (pares persona libro) cuya fecha de préstamo no sea válida.

Si se considera necesario, pueden usarse predicados predefinidos como, por ejemplo, `not`, `member` o `append`, y/o el predicado `validator`, definido en la pregunta anterior.

Ejemplo de uso.

```
?- incorrect(P,B).
P = "Juan",
B = "La broma" ;
P = "Pepe",
B = "El castillo" ;
P = "Alicia",
B = "El nombre de la rosa" ;
P = "Juan",
B = "La hija del canibal".
```

Solución:

```
incorrect(P,B) :- prestamos(DB,F), member((P,B),DB), not(validator(F)).
```