Examen de Prácticas - 25 de enero de 2021
LTP (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 `mapfilter` cuyo tipo es:

```
mapfilter :: (a -> a) -> (a -> Bool) -> [a] -> [a]
```

Dadas `g`, una función de tipo `(a -> a)`, `f`, una función de tipo `(a -> Bool)`, y `lx`, una lista de tipo `a`, `(mapfilter g f lx)` devuelve una lista con los siguientes elementos:

- Si un elemento `x` en la posición `i` de `lx` cumple que `(f x)` es `true`, entonces en esa posición `i` de la lista a devolver se sitúa el valor `(g x)`.
- En caso contrario (si el elemento `x` en la posición `i` de `lx` cumple que `(f x)` es `false`), entonces en esa posición `i` de la lista a devolver se sitúa el mismo valor `x`.

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

Ejemplos de uso.

```
*Main> let lis = [3,7,2,1,3,8,4,5,7,-1]
*Main> let g = (*2)
*Main> let f = (>3)
*Main> mapfilter g f lis
[3,14,2,1,3,16,8,10,14,-1]
```

```
*Main> let l2 = ["ab","cd","af","wq","bb","xz"]
*Main> let g2 = ('k':)
*Main> let f2 = (>"bb")
*Main> mapfilter g2 f2 l2
["ab","kcd","af","kwq","bb","kxz"]
```

Solución con listas intensionales:

```
mapfilter :: (a -> a) -> (a -> Bool) -> [a] -> [a]
mapfilter g f x = [if f y then g y else y | y <- x]
```

Solución recursiva:

```
mapfilter :: (a -> a) -> (a -> Bool) -> [a] -> [a]
mapfilter _ _ [] = []
mapfilter g f (x:xs) = y : mapfilter g f xs
   where y = if f x then g x else x
```

Pregunta 2 – Haskell (2.20 puntos)

Considera disponible la siguiente definición, correspondiente a árboles binarios, ordenados, de enteros:

```
data BinTreeInt = Void | Node Int BinTreeInt BinTreeInt deriving (Eq, Show)
```

Define una función `sumMaxPath` cuyo tipo es: `sumMaxPath :: BinTreeInt -> Int`

Dado un árbol de tipo `BinTreeInt`, la función devuelve la suma los enteros que hay en el camino desde la raíz del árbol hasta su hoja más a la derecha (el valor máximo del árbol).

Ejemplo de uso.

```
*Main> let tree = (Node 5 (Node 3 (Node 1 Void Void)(Node 4 Void Void)) (Node 6 Void
(Node 8 Void Void)))
*Main> sumMaxPath tree
19
```

Solución:

```
sumMaxPath :: BinTreeInt -> Int
sumMaxPath Void = 0
sumMaxPath (Node x _ der) = x + sumMaxPath der
```

Pregunta 3 – Haskell (2.20 puntos)

Considera disponibles las siguientes definiciones:

```
type Title = String
type Authors = [String]
type Duration = Float
type Languages = Int
data Obra = Obra Title Authors
data Audiovisual = Audiovisual Obra Duration Languages
class Translation a where
   translated :: a -> Bool
   translations :: a -> Int
```

Donde:

- `Obra` es un tipo de datos que representa obras artísticas con los siguientes valores:
  - `Title`, un string que es el título de la obra.
  - `Authors`, una lista de string que contiene los nombres de los autores de la obra.
- `Audiovisual` es un tipo de datos que representa obras artísticas audiovisuales (por ejemplo, películas), con los siguientes valores:
  - `Obra`, con el título y los autores (o directores) de la obra.
  - `Duration`, un real igual a la duración de la obra, en minutos.
  - `Languages`, un entero igual al número de idiomas disponibles en la obra.
- `Translation` es una clase de tipos que define 2 funciones para los tipos `a` que la instancien:

- `translated` que devuelve un valor lógico indicando si el dato de tipo `a` está traducido (o doblado) a otros idiomas.
- `translations` que devuelve el número de traducciones (o doblajes) del dato de tipo `a`.

Se pide: Instanciar la clase `Translation` para el tipo `Audiovisual` teniendo en cuenta que:

- `translated` ha de devolver `True` si el número de idiomas disponibles en la obra es mayor que `1`, y ha de devolver `False` en caso contrario.
- `translations` ha de devolver el número de doblajes del dato de tipo `Audiovisual`, que es el número de idiomas disponibles en la obra menos `1`.

Ejemplo de uso.

```
*Main> let directors = ["Phil Lord", "Christopher Miller"]
*Main> let movie = Audiovisual (Obra "The Lego Movie" directors) 100 3
*Main> translated movie
True
*Main> translations movie
2
```

Solución:

```
instance Translation Audiovisual where
   translated (Audiovisual _ _ l) = l > 1
   translations (Audiovisual _ _ l) = l - 1
```

Pregunta 4 – Prolog (1.70 puntos)

Dada la siguiente base de conocimiento:

```
% "Fundamentos de Algoritmia", de "G. Brassard" y "P. Bratley",
% libro publicado por "Prentice Hall" en 1997
book("Fundamentos de Algoritmia", ["G. Brassard", "P. Bratley"], "Prentice Hall", 1997).
book("Sistemas Operativos", ["William Stallings"], "Prentice Hall", 1997).
book("Fundamentos de Bases de Datos", ["H. Korth", "A. Silberschatz"], "McGrawHill", 1993).
book("Fisica Cuantica", ["Robert Eisberg", "Robert Resnick"], "Limusa", 1979).
book("Sistemas Operativos", ["Milan Milenkovic"], "McGrawHill", 1994).
```

Define un predicado `published` que permita consultar los títulos de libros publicados por una editorial dada, o bien las editoriales que han publicado un título dado, en ambos casos antes de un año dado.

Ejemplos de uso.

```
?- published(B,"McGrawHill",1994).
B = "Fundamentos de Bases de Datos" ;
false.
```

```
?- published("Sistemas Operativos",E,2000).
E = "Prentice Hall" ;
E = "McGrawHill".
```

Solución:

```
published(B, E, A) :- book(B, _, E, Y), Y < A.
```

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", "Niebla"), ("Juan", "Hamlet"), ("Pepe", "Odessa"),
           ("Alicia", "La ciudad de las bestias")], date(7,ene,2021)).
prestamos([("Pepe", "El pirata"), ("Ana", "Cumbres borrascosas"), ("Ana", "El unicornio"),
           ("Juan", "Niebla")], date(21,ene,2021)).
```

Define un predicado `obtain` que permita consultar las personas que tienen en préstamo un libro dado, o bien los libros que tiene en préstamo una persona dada, en ambos casos pudiendo especificar día, mes y año de la consulta.
Si se considera necesario, pueden usarse predicados predefinidos como, por ejemplo, `member` o `append`.

Ejemplos de uso.

```
?- obtain("Ana",B,_).
B = "Ana Karenina" ;
B = "Cumbres borrascosas" ;
B = "El unicornio".
```

```
?- obtain(P,"Niebla",date(D,ene,2021)).
P = "Alicia",
D = 7 ;
P = "Juan",
D = 21.
```

Solución:

```
obtain(P,B,date(D,M,Y)) :- prestamos(DB,date(D,M,Y)), member((P,B),DB).
```