Hoja de Ejercicios 1

Resuelve los siguientes ejercicios en Haskell (correspondientes a los temas 2 y 3 del temario de la asignatura).

Ejercicios – Primera parte

a) Implementar una función en Haskell que dados tres números enteros determine si están ordenados de menor a mayor.

b) Implementar una función en Haskell que dados tres números enteros los devuelva ordenados de menor a mayor.

c) Implementar en Haskell una función que reciba un número real y devuelva una tupla con su parte entera y sus dos primeros decimales (como número entero).

d) Crear una función que reciba el radio de una circunferencia y devuelva una 2-tupla con la longitud de la circunferencia y con el área del círculo. Emplea una definición local con la cláusula `where` para almacenar el valor de Pi (Nota: no se debe utilizar la función predefinida `pi`). A continuación crear una función con el mismo cometido empleando la definición local `let`.

e) Implementar la función predefinida de listas `concat`, que se llamará `concatenar`, utilizando la definición de listas por comprensión (no se puede utilizar recursividad).

f) Implementar una función que dado un número entero devuelva en una lista todos los factores de dicho número. Se debe utilizar la definición de listas por comprensión.

   En matemáticas, los **factores de un número** son los números enteros que pueden multiplicarse juntos para igualar ese número. O también se puede decir que los **factores de un número** son números enteros por el que un número es divisible.

g) Implementar una función que diga si un número es primo. Para ello se debe utilizar la función que calcula el número de factores de un número (ejercicio f).

   Nota: Si para resolver el ejercicio se deben comparar dos listas, se puede hacer con el operador de igualdad de listas (==). Por ejemplo:

```
> [1,2,3] == [1,2,3]
True
> [1,2,3] == [1,2]
False
```

h) Implementar una función que diga cuántos caracteres en mayúscula están contenidos en una frase dada. Se deberá utilizar la definición de listas por comprensión.

Ejercicios – Segunda parte

i) Implementar una función que dada una tupla de tres elementos, donde cada uno de ellos es a su vez una tupla de dos elementos de tipo `String` e `Int` respectivamente, retorne el primer elemento de cada tupla interna. Se deberá utilizar ajuste de patrones.

j) Implementar una función que devuelve `True` si la suma de los cuatro primeros elementos de una lista de números enteros es un valor menor a 10 y devolverá `False` en caso contrario. Se deberá utilizar ajuste de patrones.

k) Implementar una función que dado un carácter, que representa un punto cardinal, devuelva su descripción. Por ejemplo, dado `'N'` devuelva `"Norte"`.

l) Implementar una función que dada una frase retorne un mensaje donde se indique cuál es la primera y última letra de la frase original. Un ejemplo de aplicación de la función podría ser:

```
> procesarFrase "El perro de San Roque"
"La primera letra de la frase ''El perro de San Roque'' es 'E' y  la
ultima letra es 'e'"
```

Nota: No se permite el uso de recursividad. Se debe usar ajuste de patrones y se puede utilizar también patrones nombrados (para referirse a la cadena de entrada).

m) Implementar una función que dado un número entero devuelva mensajes indicando en qué rango de valores se encuentra dicho número (menor de 10, entre 10 y 20 o mayor de 20). Se debe utilizar definiciones locales.

Ejemplos de aplicación de la función son:

```
> clasificarValorEntrada 20
"El valor de entrada es mayor o igual a 10 y menor o igual a 20"

> clasificarValorEntrada 9
"El valor de entrada es menor que 10"

> clasificarValorEntrada 35
"El valor de entrada es mayor que 20"
```

Pista: La cadena “El valor de entrada” se repite constantemente, por ello una definición local tiene sentido para que sólo se defina una vez.

n) Implementar una función que dada una cadena de caracteres y un carácter, indique el número de apariciones del carácter en la cadena. No se debe utilizar recursividad, sí ajuste de patrones. Pista: utilizar la definición de listas por comprensión.

Ejemplos de aplicación de la función:

```
> contarApariciones "casa" 'c'
1

> contarApariciones "casa" 'a'
2

> contarApariciones "" 'c'
0
```