Trabajo Práctico Integrador Nro 2.

Unidad 4: Paradigma Funcional.

Objetivos:

El objetivo del presente trabajo práctico integrador es la evaluación de los temas de la unidad número 4: Paradigma de Programación Funcional, que se detallan a continuación: definición de funciones; uso de expresiones en Haskell tales como guardas, if – then – else, case of, otherwise, where, entre otras; listas por comprensión; recursividad; tuplas.

Enunciado:

A continuación, se presentan 6 consignas que usted deberá resolver y codificar utilizando el lenguaje Haskell bajo el entorno WinHugs o hugs para Linux o mac. Para resolver cada una de estas consignas, se podrá implementar una o varias funciones de acuerdo con lo solicitado y/o según lo considere necesario. Se deberá respetar en la codificación que se realice para resolver cada consigna, el o los tipos de expresiones en Haskell y/o los mecanismos de resolución de problemas según se especifique.

Consigna 1) Implementar una función en Haskell que reciba dos parámetros: el primer parámetro deberá ser un número entero **x**, y el segundo parámetro deberá ser otro número entero **n**. La función deberá devolver otro número entero, de acuerdo a los siguientes criterios:

- En caso de que el segundo parámetro **n**, sea un número **mayor que 0**, la función deberá devolver el **siguiente número consecutivo** respecto el primer parámetro **x**, es decir, el número **x + 1**.
- En caso de que el segundo parámetro **n**, sea un número **menor que 0**, la función deberá devolver el **anterior número consecutivo** respecto el primer parámetro **x**, es decir, el número **x - 1**.
- En caso de que el segundo parámetro **n**, sea un número **igual que 0**, la función deberá devolver directamente el valor **0**.

Ejemplo 1:
```
Main> funcion1 12 5
      13
```
Ejemplo 2:
```
Main> funcion1 12 (-2)
      11
```
Ejemplo 3:
```
Main> funcion1 6 (-9)
      5
```
Ejemplo 4:
```
Main> funcion1 0 3
      1
```
Ejemplo 5:
```
Main> funcion1 2 0
      0
```

Consigna 2) Implementar una función en Haskell que reciba 2 parámetros: el primer parámetro deberá ser una lista de números enteros, y el segundo parámetro deberá ser únicamente un número entero **n**. Y devuelva una lista de números enteros, de acuerdo con los siguientes criterios.

- En caso de que el segundo parámetro **n**, sea un número entero **mayor que 0**, la función deberá generar otra lista en la que cada elemento sea el **siguiente número consecutivo** respecto a cada número **x** de la lista recibida como primer parámetro. Obviamente deberá tener la misma cantidad de elementos que tiene la lista que se recibe como primer parámetro.
- En caso de que el segundo parámetro **n**, sea un número entero **menor que 0**, la función deberá generar otra lista en la que cada elemento sea el **anterior número consecutivo** respecto a cada número **x** de la lista recibida como primer parámetro. Obviamente deberá tener la misma cantidad de elementos que tiene la lista que se recibe como primer parámetro.
- En caso de que el segundo parámetro **n**, sea un número **igual que 0**, la función deberá generar una lista conformada solo por elementos 0, tantos como elementos tenga la lista recibida como primer parámetro.

Se deberá implementar aplicando recursividad, y reutilizando la función implementada con la que se resolvió la consigna 1).

Ejemplo 1:
```
Main> funcion2 [10,12,9,3,4,1] 1
[11,13,10,4,5,2]
```
Ejemplo 2:
```
Main> funcion2 [10,12,9,3,4,1] 0
[0,0,0,0,0,0]
```
Ejemplo 3:
```
Main> funcion2 [10,12,9,3,4,1] (-3)
[9,11,8,2,3,0]
```
Ejemplo 4:
```
Main> funcion2 [1,2,3] 9
[2,3,4]
```
Ejemplo 5:
```
Main> funcion2 [1,2,3] (-8)
[0,1,2]
```

Consigna 3) Implementar una función que calcule el valor de $\mathbf{e^x}$, siendo x un parámetro de la función. Para ello utilizar el desarrollo de la serie de Taylor para calcular las potencias de **e**:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Dado que la serie plantea una sumatoria para n desde 0 hasta infinito, la función a desarrollar requerirá como parámetro además del exponente x, la cantidad de términos de la serie, la cual operará como valor máximo de n.

Para resolver esta consigna debe utilizarse recursividad.

Ejemplo 1:
```
Main> funcion3 1 1
 2.0
```

Ejemplo 2:
```
Main> funcion3 1 10
2.718282
```
Ejemplo 3:
```
Main> funcion3 2 10
7.388995
```
Ejemplo 4:
```
Main> funcion3 10 20
21991.48
```
Ejemplo 5:
```
Main> funcion3 10 30
22026.46
```
Ejemplo 6:
```
Main> funcion3 0 30
1.0
```

Consigna 4) Implementar una función que calcule el seno de un ángulo expresado en radianes. La función debe recibir como parámetro un ángulo expresado en radianes para el cual se quiere calcular el seno. Para el cálculo debe utilizarse la serie de Taylor correspondiente, sumando siempre los primeros 10 términos de la misma.

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

La función debe retornar el valor de sin(x) para la x recibida como parámetro. Si el valor de x no se encuentra entre 0 y 6.284 debe retornarse -2 como resultado de la misma.

Ejemplo 1:
```
Main> funcion4 3.14
0.001592378
```
Ejemplo 2:
```
Main> funcion4 0
0.0
```
Ejemplo 3:
```
Main> funcion4 3
0.1411201
```
Ejemplo 4:
```
Main> funcion4 6.3
-2.0
```
Ejemplo 5:
```
Main> funcion4 5.5
-0.7055353
```
Ejemplo 6:
```
Main> funcion4 (-0.5)
2.0
```

Consigna 5) Implementar una función que calcule el seno de una lista de ángulos expresados en radianes. La función debe recibir como parámetro una lista de números conteniendo los distintos ángulos para los que se quiere calcular el seno. Para el cálculo debe utilizarse la serie de Taylor implementada para resolver la consigna 4), sumando siempre los primeros 10 términos de esta.
La función debe retornar una lista conteniendo una tupla por cada ángulo de la lista de entrada, cada tupla debe contener 2 elementos: el ángulo recibido en la lista de entrada como primer elemento y el valor de sin(x) como segundo elemento; para todas las x pertenecientes a la lista recibida como parámetro. Si alguno de los valores de x no se encuentra entre 0 y 6.284 debe retornarse -2 como resultado para dicha x.
Para resolver esta consigna debe utilizarse listas por comprensión.

Ejemplo 1:
```
Main> funcion5 []
[]
```
Ejemplo 2:
```
Main> funcion5 [3.14]
[(3.14,0.001592378)]
```
Ejemplo 3:
```
Main> funcion5 [3.14,0,0,0]
[(3.14,0.001592378),(9,0.0),(0,0.0),(0,0.0)]
```
Ejemplo 4:
```
Main> funcion5 [3.14,0,0,0,1.57]
[(3.14,0.001592378),(0,0.0),(0,0.0),(,0.0),(1.57,0.9999996)]
```
Ejemplo 5:
```
Main> funcion5 [3.14,50,-1,1.57]
[(3.14,0.001592378),(50,-2.0),(-1,2.0),(1.57,0.9999996)]
```
Ejemplo 6:
```
Main> funcion5 [-1,-1]
[(-1,-2.0),(-1,2.0)]
```

Consigna 6) Implementar una función en Haskell que reciba dos parámetros: el primer parámetro deberá ser un número entero **x**, y el segundo parámetro deberá ser otro número entero **n**. La función deberá devolver una **tupla de 2 elementos**, de acuerdo con los siguientes criterios:

- En caso de que el segundo parámetro **n**, sea un número **mayor que 0**, la función deberá devolver una tupla conformada por el número entero **x** y el **siguiente número consecutivo** a **x**, es decir: **(x, x+1)**.
- En caso de que el segundo parámetro **n**, sea un número **menor que 0**, la función deberá devolver una tupla conformada por el número entero **x** y el **anterior número consecutivo** a **x**, es decir: **(x, x-1)**.
- En caso de que el segundo parámetro **n**, sea un número **igual que 0**, la función deberá devolver una tupla conformada por dos elementos iguales a 0, es decir: **(0, 0)**.

Ejemplo 1:
```
Main> funcion6 2 1
(2,3)
```
Ejemplo 2:
```
Main> funcion6 2 0
(0,0)
```
Ejemplo 3:
```
Main> funcion6 2 (-1)
(2,1)
```
Ejemplo 4:
```
Main> funcion6 5 (-3)
(5,4)
```
Ejemplo 5:
```
Main> funcion6 5 3
(5,6)
```
Ejemplo 6:
```
Main> funcion6 0 3
(0,1)
```

Tabla de valoración de los ítems evaluados

Nro. de ítem	Ítems o requerimiento a evaluar	Puntaje	Observaciones	Obtenido
1	Definición correcta de la función solicitada: argumentos y tipo de retorno. Utilización apropiada de expresiones y funciones provistas por el lenguaje.	15		
2	Definición correcta de la función solicitada: argumentos y tipo de retorno. Utilización apropiada de expresiones y funciones provistas por el lenguaje. Reutilización de funciones. Listas y recursividad.	15		
3	Definición correcta de la función solicitada: argumentos y tipo de retorno. Utilización apropiada de expresiones y funciones provistas por el lenguaje. Listas y recursividad.	20		
4	Definición correcta de la función solicitada: argumentos y tipo de retorno. Utilización apropiada de expresiones y funciones provistas por el lenguaje. Listas y recursividad.	20		
5	Definición correcta de la función solicitada: argumentos y tipo de retorno. Reutilización de funciones. Utilización de listas por comprensión.	15		
6	Definición correcta de la función solicitada: argumentos y tipo de retorno. Utilización correcta de expresiones y funciones provistas por el lenguaje. Reutilización. Retorno de tuplas.	15		
	Total	**100**		

Condiciones de entrega:

- Este trabajo práctico integrador **se deberá realizar en forma grupal**.
- Deberán nombrar el **archivo comprimido** con el siguiente formato: TP2_Apellido1erIntegranteNroLeg1erIntegrante_Apellido2doIntegrenteNroLeg2doIntegrante_Apellido3erIntegranteNroLeg3erIntegrante. El orden en el cual deberán colocar los legajos y apellidos cada integrante en el nombre del archivo comprimido será de acuerdo al orden alfabético de los apellidos de los integrantes, desde la A hasta la Z.
- Se deberá subir una carpeta comprimida que contenga en su interior el archivo.hs correspondiente a la codificación en Haskell, además debe contener un archivo .txt con las invocaciones y resultados de las funciones solicitadas para hacer funcionar el programa.
- Plazo máximo de entrega de este trabajo: **HASTA el 15/10/2021 a las 23:55 hs.**
- Todas las consultas de este TP2 pueden realizarlas a través del foro del aula virtual o en las clases de tutoría dispuestas para tal fin.