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. 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 las siguientes funciones en Haskell:

1. a) Una función llamada **es_vocal**, que reciba como único parámetro un carácter **car** y devuelva True en caso de que **car** sea una vocal (`'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'`), o False, en caso contrario.

1. b) Una función llamada **es_digito**, que reciba como único parámetro un carácter **car** y devuelva True en caso de que **car** sea un dígito (`'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'`), o False, en caso contrario.

Ejemplo 1:
```
Main> es_vocal 'a'
      True
```
Ejemplo 2:
```
Main> es_vocal '2'
      False
```
Ejemplo 3:
```
Main> es_digito 'a'
      False
```
Ejemplo 4:
```
Main> es_digito 'b'
      False
```
Ejemplo 5:
```
Main> es_vocal 'b'
      False
```
Ejemplo 6:
```
Main> es_digito '1'
      True
```

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

- En caso de que el segundo parámetro **car**, sea una vocal (`'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'`), la función deberá devolver el **doble del número entero recibido como primer parámetro**, es decir, el número **2 * x**.
- En caso de que el segundo parámetro **car**, sea un dígito (`'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'`), la función deberá devolver el **triple del número entero recibido como primer parámetro**, es decir, el número **3 * x**.
- En caso de que el segundo parámetro **car**, no sea una vocal (`'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'`), ni tampoco un dígito (`'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'`), la función deberá devolver **el mismo entero recibido como primer parámetro**, es decir, el número **x**.

Ejemplo 1:
```
Main> funcion2 12 'a'
  24
```
Ejemplo 2:
```
Main> funcion2 12 'b'
  12
```
Ejemplo 3:
```
Main> funcion2 6 '2'
  18
```
Ejemplo 4:
```
Main> funcion2 0 'á'
  0
```
Ejemplo 5:
```
Main> funcion2 2 'e'
  4
```

Consigna 3) 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 *carácter* **car**. Y devuelva una *lista de números enteros*, de acuerdo con los siguientes criterios.

- En caso de que el segundo parámetro **car**, sea una vocal (`'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'`), la función deberá devolver otra lista en la que cada elemento entero sea el **doble de cada uno de los números enteros recibidos en la lista como primer parámetro**.
- En caso de que el segundo parámetro **car**, sea un dígito (`'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'`), la función deberá devolver otra lista en la que cada elemento entero sea el **triple de cada uno de los números enteros recibidos en la lista como primer parámetro**.
- En caso de que el segundo parámetro **car**, no sea una vocal (`'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'`), ni tampoco un dígito (`'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'`), la función deberá devolver otra lista en la que cada elemento entero sea el **mismo elemento de la lista recibida como primer parámetro**.

Se deberá implementar aplicando recursividad, y reutilizando las funciones ya implementadas que considere necesarias.

Ejemplo 1:
```
Main> funcion3 [10,12,9,3,4,1] 'a'
[20,24,18,6,8,2]
```
Ejemplo 2:
```
Main> funcion3 [10,12,9,3,4,1] 'b'
[10,12,9,3,4,1]
```
Ejemplo 3:
```
Main> funcion3 [10,12,9,3,4,1] '2'
[30,36,27,9,12,3]
```
Ejemplo 4:
```
Main> funcion3 [1,2,3] 'á'
[1,2,3]
```
Ejemplo 5:
```
Main> funcion3 [1,2,3] 'e'
[2,4,6]
```

Consigna 4) Implementar una función en Haskell que reciba 3 parámetros: el primer parámetro deberá ser una *lista de números enteros*, el segundo parámetro deberá ser únicamente un *carácter* **car,** y el tercer parámetro sea un número *entero* **cant**. Y devuelva una lista que se genere tomando sólo los primeros cant elementos de la lista que se recibe como primer parámetro, de acuerdo con los siguientes criterios.

- En caso de que el segundo parámetro **car**, sea una vocal (`'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'`), la función deberá devolver otra lista de hasta **cant** elementos, de tal manera que cada elemento entero sea el **doble de cada uno de los cant primeros elementos de la lista recibida como primer parámetro.** Si **cant** supera la cantidad de elementos de la lista recibida como primer parámetro, se considerarán la cantidad total de elementos de esta lista (cantidad de elementos de la lista recibida como primer parámetro).
- En caso de que el segundo parámetro **car**, sea un dígito (`'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'`), la función deberá devolver otra lista de hasta **cant** elementos, de tal manera que cada elemento entero sea el **triple de cada uno de los cant primeros elementos de la lista recibida como primer parámetro.** Si **cant** supera la cantidad de elementos de la lista recibida como primer parámetro, se considerarán la cantidad total de elementos de esta lista (cantidad de elementos de la lista recibida como primer parámetro).
- En caso de que el segundo parámetro **car**, no sea una vocal (`'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'`), ni tampoco un dígito (`'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'`), la función deberá devolver otra lista de hasta **cant** elementos, de tal manera que cada elemento entero sea **exactamente igual a cada uno de los cant primeros elementos de la lista recibida como primer parámetro.** Si **cant** supera la cantidad de elementos de la lista recibida como primer parámetro, se considerarán la cantidad total de elementos de esta lista (cantidad de elementos de la lista recibida como primer parámetro).

Se deberá implementar aplicando recursividad, y reutilizando las funciones que considere necesarias.

Ejemplo 1:
```
Main> funcion4 [10,12,9,3,4,1] 'a' 3
[20,24,18]
```
Ejemplo 2:
```
Main> funcion4 [10,12,9,3,4,1] 'b' 2
[10,12]
```
Ejemplo 3:
```
Main> funcion4 [10,12,9,3,4,1] '2' 10
[30,36,27,9,12,3]
```
Ejemplo 4:
```
Main> funcion4 [1,2,3] 'á' 1
[1]
```
Ejemplo 5:
```
Main> funcion4 [1,2,3] 'e' 0
[]
```

Consigna 5) Implementar una función que reciba como parámetro un número *entero* y devuelva una *tupla de 2 elementos enteros*, cuyo primer elemento sea el mismo número entero recibido como parámetro y el segundo elemento de la tupla sea el triple del primer elemento.

Ejemplo 1:
```
Main> funcion5 2
(2,6)
```
Ejemplo 2:
```
Main> funcion5 3
(3,9)
```
Ejemplo 3:
```
Main> funcion5 0
(0,0)
```
Ejemplo 4:
```
Main> funcion5 10
(10,30)
```

Consigna 6) Implementar una función que reciba como parámetro *2 números enteros*: **n** y **cant**, y devuelva una *lista con **cant** tuplas de 2 elementos enteros*. La primera tupla de esta lista deberá estar conformada de la siguiente forma: el primer elemento de esta tupla deberá ser el mismo número **n** correspondiente al primer parámetro recibido en la función, y el segundo elemento de esta primera tupla deberá ser el triple del primer elemento de esta tupla. Cada una de las siguientes (n-1) tuplas deberán estar conformadas de la siguiente forma: el primer elemento deberá ser igual al segundo elemento de la anterior tupla, y el segundo elemento de esta tupla deberá ser igual al triple del primer elemento de la actual tupla, y así sucesivamente.

Ejemplo 1:
```
Main> funcion6 2 3
[(2,6),(6,18),(18,54)]
```
Ejemplo 2:
```
Main> funcion6 2 0
[]
```
Ejemplo 3:
```
Main> funcion6 3 (-1)
[]
```
Ejemplo 4:
```
Main> funcion6 0 2
[(0,0),(0,0)]
```
Ejemplo 5:
```
Main> funcion6 10 5
[(10,30),(30,90),(90,270),(270,810),(810,2430)]
```

Tabla de valoración de los ítems evaluados

Nro. de ítem	Ítems o requerimientos 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.	15		
3	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.	20		
4	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.	20		
5	Definición correcta de la función solicitada: argumentos y tipo de retorno. Reutilización de funciones. Utilización de tuplas.	15		
6	Definición correcta de la función solicitada: argumentos y tipo de retorno. Reutilización de funciones. Utilización de lista 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: TP1_NroLeg1erIntegranteApellido1erIntegrante_NroLeg2doIntegranteApellido2doIntegrente_NroLeg3erIntegranteApellido3erIntegrante. El orden en el cual deberán colocar los legajos y apellidos cada integrante en el nombre del archivo comprimido será de acuerdo con el 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: lo especificará cada comisión de acuerdo con su propio cronograma de clases.
- Todas las consultas de este TP2 pueden realizarlas a través del foro del aula virtual.

Instancia de recuperación del trabajo práctico:

En caso de recuperar el trabajo práctico nro. 2 de programación funcional, se deberán presentar todas las funciones correspondientes a la instancia de entrega original, funcionando correctamente, más las funciones correspondientes a las siguientes consignas:

Consigna 7) Implementar una función que reciba como parámetro un *número entero* y devuelva una *tupla de 3 elementos enteros*, cuyo primer elemento sea el mismo número entero recibido como parámetro, el segundo elemento de la tupla sea el doble del primer elemento, y el tercer elemento sea el doble del segundo elemento de dicha tupla.

Ejemplo 1:
```
Main> funcion7 2
(2,4,8)
```
Ejemplo 2:
```
Main> funcion7 3
(3,6,12)
```
Ejemplo 3:
```
Main> funcion7 0
(0,0,0)
```
Ejemplo 4:
```
Main> funcion7 10
(10,20,40)
```

Consigna 8) Implementar una función que reciba como parámetro *2 números enteros*: **n** y **cant**, y devuelva una *lista con **cant** tuplas de 3 elementos enteros*. La primera tupla de esta lista deberá estar conformada de la siguiente forma: el primer elemento de esta tupla deberá ser el mismo número **n** correspondiente al primer parámetro recibido en la función, el segundo elemento de esta primera tupla deberá ser el doble del primer elemento de esta tupla, y el tercer elemento de esta tupla deberá ser el doble del segundo elemento de dicha tupla. Cada una de las siguientes (n-1) tuplas deberán estar conformadas de la siguiente forma: el primer elemento deberá ser igual al último elemento de la anterior tupla, y el segundo elemento de esta tupla deberá ser igual al doble del primer elemento de la actual tupla, y el tercer elemento de esta tupla deberá ser igual al doble del segundo elemento de dicha tupla, y así sucesivamente.

Ejemplo 1:
```
Main> funcion8 2 3
[(2,4,8),(8,16,32),(32,64,128)]
```
Ejemplo 2:
```
Main> funcion8 2 0
[]
```
Ejemplo 3:
```
Main> funcion8 3 (-1)
[]
```
Ejemplo 4:
```
Main> funcion8 0 2
[(0,0,0),(0,0,0)]
```
Ejemplo 5:
```
Main> funcion8 10 2
[(10,20,40),(40,80,160)]
```

Tabla de valoración de los ítems evaluados en la instancia de recuperación

Es condición necesaria para aprobar la instancia de recuperación que se presenten todas las funciones correspondientes a la instancia original correctamente implementadas.

Nro. de consigna	Ítems o requerimientos a evaluar	Puntaje	Observaciones	Obtenido
1 a 6	Implementación correcta de todas las funciones correspondientes a la instancia original. Definición correcta de la función solicitada: argumentos y tipo de retorno. Reutilización de funciones. Utilización de tuplas.	45		
7	Definición correcta de la función solicitada: argumentos y tipo de retorno. Reutilización de funciones. Utilización de tuplas.	30		
8	Definición correcta de la función solicitada: argumentos y tipo de retorno. Reutilización de funciones. Utilización de listas de tuplas.	25		
	Total	**100**		