**Practica 2**

**Martin Valencia Vallejo y Daniel Montoya Arenas**

Delivery of the solution is done in two ways:

I. Through the site set up in the OnlineGDB Classroom by clicking on this link. Each member of the team must enter the link and deliver the proposed solution.

II. Include a text document with the names of the two team members and the answers to the following points:

1. Describe in detail the process you carried out to solve the exercise.

2. Write a form in which the currying process is manifested in the proposed solution.

3. For each function developed, indicate whether you identify it as polymorphic, of higher order, or, if you do not consider it to be classified in one or both criteria, justifying each defined determination.

4. Store the developed source file(s) in a GitHub repository, invite the teacher as a collaborator and place a link to access the developed repository at this point in the written document.

5. For the presentation of the developed solution, the team must elaborate a video in which each member appears detailing step by step the process that was carried out, all team members must appear in the video and participate. The video must be uploaded to the Internet and a link must be generated so that it can be accessed for the corresponding review of the delivery. The duration of the video should not exceed seven (7) minutes. This link must be included in this point of the document.

**Solución:**

**1.**

A continuación, se detallará el funcionamiento de las funciones clave y el porqué de las operaciones matemáticas utilizadas.

Extracción de Partes del Número

La descomposición del número en partes significativas se realiza con tres funciones principales:

1. \*tomarTresPrim\*

tomarTresPrim :: Double -> Int

tomarTresPrim x = floor (x / 100000)

Esta función extrae los tres primeros dígitos del número. Al dividir el número por 100000 y tomar el piso del resultado, se eliminan los dígitos menos significativos, dejando sólo los tres primeros. Esto es crucial para interpretar el periodo académico.

2. \*tomarDos\*

tomarDos :: Int -> Int

tomarDos x = mod (div x 1000) 100

Aquí, se extraen los dos dígitos centrales que representan el programa académico. Primero, se divide el número por 1000 para descartar los tres últimos dígitos. Luego, se aplica el módulo 100 para obtener los dos últimos dígitos del resultado anterior. Esta combinación de división y módulo es esencial para aislar los dígitos deseados.

3. \*tomarTresUlt\*

tomarTresUlt :: Double -> Int

tomarTresUlt x = round x - ((floor (x / 1000)) \* 1000)

Esta función obtiene los últimos tres dígitos del número. Se redondea el número y se resta el resultado de dividir el número por 1000 (despreciando los decimales) multiplicado por 1000. Esta resta elimina todos los dígitos menos los tres últimos.

Análisis de Divisores y Clasificación del Programa

La clasificación del programa académico se basa en las propiedades matemáticas de los divisores del número. Esto se realiza a través de dos funciones:

1. \*buscarDivisores\*

buscarDivisores :: Int -> [Int]

buscarDivisores x = divisoresRec x x []

Esta función inicia la búsqueda de todos los divisores de un número utilizando una función recursiva divisoresRec.

2. \*divisoresRec\*

divisoresRec :: Int -> Int -> [Int] -> [Int]

divisoresRec x y xs

| y == 1 = 1 : xs

| mod x y == 0 = divisoresRec x (y - 1) (y : xs)

| otherwise = divisoresRec x (y - 1) xs

La función divisoresRec es recursiva y encuentra todos los divisores de x desde y hasta 1. Si y es un divisor de x (esto se verifica con mod x y == 0), y se añade a la lista de divisores y la recursión continúa con y - 1. Si no es un divisor, simplemente se continúa con y - 1. La recursión finaliza cuando y llega a 1, asegurando que todos los divisores posibles han sido considerados.

1. \*identificar2\*

identificar2 :: [Int] -> String

identificar2 xs

| sum1 == last xs = "Engineering"

| sum1 < last xs = "Humanities"

| sum1 > last xs = "Administrative"

where sum1 = sum (init xs)

Una vez que se tiene la lista de divisores, identificar2 clasifica el programa académico basado en la suma de todos los divisores excepto el número mismo (sum (init xs)). Si la suma es igual al número, el programa es de "Engineering" (número perfecto). Si la suma es menor, es de "Humanities" (número deficiente). Si la suma es mayor, es de "Administrative" (número abundante). Este análisis se fundamenta en propiedades matemáticas de los números y proporciona una clasificación precisa basada en la teoría de números.

Verificación y Completado de Información

Para asegurar que el número procesado tenga siempre 8 dígitos, se implementan funciones que completan los números si tienen menos dígitos:

1. \*completarProgramaAcademico\*

completarProgramaAcademico :: Double -> Int

completarProgramaAcademico y = (tresPrim \* 100000) + (mitad \* 1000) + tresUlt

where

tresPrim = floor (y / 1000)

tresUlt = round y - (floor (y / 1000) \* 1000)

mitad = mod (floor (y / 1000)) 10

Esta función asegura que el número del programa académico tenga el formato adecuado rellenando los dígitos necesarios. Los tres primeros dígitos (tresPrim) se obtienen dividiendo entre 1000, los tres últimos (tresUlt) se obtienen mediante una combinación de redondeo y resta, y el dígito del medio (mitad) se extrae usando el módulo 10 sobre el valor obtenido.

1. \*completarNumeroAdmisiones1\* y \*completarNumeroAdmisiones2\*

completarNumeroAdmisiones1 :: Int -> Int

completarNumeroAdmisiones1 y = (tresPrim \* 100000) + (mitad \* 1000) + ultNum

where

tresPrim = div y 1000

ultNum = mod y 10

mitad = div (mod y 1000) 10

completarNumeroAdmisiones2 :: Int -> Int

completarNumeroAdmisiones2 y = (tresPrim \* 100000) + (mitad \* 1000) + dosUlt

where

tresPrim = div y 10000

dosUlt = mod y 100

mitad = mod (div y 100) 100

Estas funciones ajustan el número de admisiones para que siempre tenga 8 dígitos. completarNumeroAdmisiones1 se usa para números menores a un millón, mientras que completarNumeroAdmisiones2 maneja números menores a diez millones. Ambas funciones dividen y modulan el número para extraer y colocar los dígitos adecuadamente en base a la cantidad de ceros que deban agregarse.

Conversión y Verificación Final

El programa finaliza con funciones que convierten y verifican los datos:

1. \*doubleToInt\* y \*enteroADouble\*

doubleToInt :: Double -> Int

doubleToInt x = floor x

enteroADouble :: Int -> Double

enteroADouble x = fromIntegral x

Estas funciones manejan la conversión entre Double e Int, garantizando la precisión en las operaciones aritméticas.

2. \*convertirAno\*

convertirAno :: Int -> Bool -> Int

convertirAno x bool

| bool = x `mod` 10

| otherwise = div x 10

Esta función convierte los tres primeros dígitos en un formato de año legible, separando el último dígito del año del resto. Si el booleano es True, se toma el último dígito (mod 10); si es False, se toman los dos primeros (div 10).

3. \*verificarPar\*

verificarPar :: Int -> String

verificarPar x

| even x = "even"

| otherwise = "odd"

Esta función determina si el número de admisiones es par o impar, utilizando la función even de Haskell.

Salida Final

La función principal main orquesta todas estas operaciones. Lee el número de entrada, lo procesa y genera una salida formateada que incluye el periodo académico, la clasificación del programa académico y el número de admisiones con su paridad. Esta función garantiza que cada parte del número se interprete y transforme correctamente, proporcionando una salida coherente y útil.

**2.**

El programa no presenta currificacion explícitamente sin embargo se presentan varias múltiples funciones compuestas tal que estas necesitan únicamente un argumento como parámetro subdiviendo múltiples tareas en diferentes funciones un ejemplo de esto podría ser la función:

identificar1 :: Int -> String

identificar1 x = identificar2 (buscarDivisores (tomarDos x))

**3.**

**tomarTresUlt:**

Polimórfica: No es polimórfica. Esta función toma un argumento de tipo Double y devuelve un resultado de tipo Int. No hay parámetros genéricos ni uso de tipos polimórficos.

Orden superior: Tampoco es de orden superior. No toma funciones como argumentos ni devuelve funciones.

**tomarTresPrim:**

Polimórfica: Al igual que la función anterior, no es polimórfica. Toma un Double y devuelve un Int.

Orden superior: No es de orden superior. No involucra funciones de alto nivel.

**tomarDos:**

Polimórfica: No es polimórfica. Toma un argumento de tipo Int y devuelve un resultado de tipo Int.

Orden superior: No es de orden superior. No utiliza funciones de alto nivel.

**buscarDivisores:**

Polimórfica: No es polimórfica. Toma un Int y devuelve una lista de Int.

Orden superior: No es de orden superior. No utiliza funciones de alto nivel.

**divisoresRec**:

Polimórfica: Tampoco es polimórfica. Toma tres argumentos de tipo Int y devuelve una lista de Int.

Orden superior: En esta función es en donde más potencial hay para usar funciones de orden superior, por ejemplo una función map

**identificar2:**

Polimórfica: No es polimórfica. Toma una lista de Int y devuelve un String.

Orden superior: No es de orden superior. No utiliza funciones de alto nivel.

**doubleToInt:**

Polimórfica: No es polimórfica. Toma un Double y devuelve un Int.

Orden superior: No es de orden superior. No involucra funciones de alto nivel.

**enteroADouble:**

Polimórfica: No es polimórfica. Toma un Int y devuelve un Double.

Orden superior: No es de orden superior. No utiliza funciones de alto nivel.

**completarProgramaAcademico:**

Polimórfica: No es polimórfica. Toma un Double y devuelve un Int.

Orden superior: No es de orden superior. No involucra funciones de alto nivel.

**verificarProgramaAcademico:**

Polimórfica: No es polimórfica. Toma un Double y devuelve un Int.

Orden superior: No es de orden superior. No utiliza funciones de alto nivel.

**completarNumeroAdmisiones1:**

Polimórfica: No es polimórfica. Toma un Int y devuelve un Int.

Orden superior: No es de orden superior. No involucra funciones de alto nivel.

**completarNumeroAdmisiones2:**

Polimórfica: No es polimórfica. Toma un Int y devuelve un Int.

Orden superior: No es de orden superior. No utiliza funciones de alto nivel.

**completarNumeroAdmisiones:**

Polimórfica: Esta es polimórfica parcialmente ya que ella acepta cualquier valor, pero a las funciones internas no

Orden superior: No es de orden superior. No involucra funciones de alto nivel.

**identificar1:**

Polimórfica: No es polimórfica. Toma un Int y devuelve un String.

Orden superior: No es de orden superior. No utiliza funciones de alto nivel.

**verificarPar:**

Polimórfica: No es polimórfica. Toma un Int y devuelve un String.

Orden superior: No es de orden superior. No involucra funciones de alto nivel.

A pesar de que no presentamos con múltiples funciones que sean de orden superior si usamos en múltiples ocasiones funciones que reciben como parámetro el resultado de otra función directamente

**4. Link al repositorio:**

[Daniel-montoya283/Practica2\_Lenguales\_Programacion (github.com)](https://github.com/Daniel-montoya283/Practica2_Lenguales_Programacion)

**Link al video:**

https://youtu.be/9DpHZBE7bc0