PRACTICE 2

Student’s ID card

Universidad EAFIT
Systems Engineering
ST0244 Programming Languages
Diego Iván Cruz Ordiéres
Diego Gonzalez & Juan Avendaño

Proceso de solución del ejercicio

Durante cualquier ejercicio de computación, es importante dar una mirada profunda al problema que se necesita resolver. Dado esto, se identificaron cuatro tareas principales dentro del ejercicio.

La primera tarea se refiere a la necesidad de acceder a los elementos de una cadena, proporcionando un índice y el número de elementos que se desea abarcar. Para esto, se plantearon dos funciones que permitían solventar dicha necesidad (una para obtener los ‘n’ primeros elementos de una lista, y otra para borrar los ‘n’ primeros elementos de una lista.

La segunda tarea identificada fue el requerimiento de determinar el periodo de ingreso del estudiante. Para ello, se desarrolló una función que, dado un string de tres caracteres numéricos, concatenara los dos primeros como un año y el último como el ciclo de ingreso. El resultado siempre tendrá por defecto el prefijo “20”. Por ejemplo, para la entrada ‘241’, el resultado sería ‘2024-1’.

La tercera tarea consistía en determinar la escuela a la que pertenecía un estudiante. Se implementaron funciones para obtener una lista con los divisores de un número dado, luego se definió una función que permitiera sumar dichos divisores y entregara dicha suma. Por último, se creó una función que decidiera cuál es la escuela del estudiante según la suma proporcionada por la función anterior (usando los criterios de numero abundante, perfecto y deficiente).

Finalmente, el último paso fue determinar la naturaleza de los últimos tres dígitos del ID del estudiante. Primero, se realizó una función que añadiera ceros a la izquierda en el caso de que la longitud de estos últimos dígitos fuera inferior a tres. Se utilizó un proceso recursivo para agregar ceros a la izquierda hasta que la cadena tuviera una longitud igual a tres. Luego, se definieron dos funciones: una que devolviera un string 'num' seguido del número obtenido (a la hora de convertir la cadena a entero se eliminan los ceros a la izquierda) y otra que devolviera 'odd' o 'even' dependiendo de si el número era impar o par, respectivamente.

Currificación

En general, la currificación se manifiesta en todo el código debido a la naturaleza de Haskell como un lenguaje funcional. En Haskell, todas las funciones son currificadas por defecto, lo que significa que, cuando definimos una función con múltiples parámetros, en realidad estamos definiendo una serie de funciones que aceptan un solo parámetro y devuelven otra función que toma el siguiente parámetro.

En nuestra práctica, la currificación se refleja en la forma en que se definen y utilizan las funciones. Cada función toma solo un argumento y devuelve otra función que espera el siguiente argumento, lo que permite aplicar parcialmente las funciones en cada paso. Esto hace que las funciones sean más flexibles y reutilizables en diferentes contextos.

Descripción de las funciones

En la solución se implementaron un total de doce (12) funciones, las cuales serán listadas a continuación.

- **numAString**: Permite llevar un numero a un String usando la funcion ‘show’. Se clasificó esta funcion como polimórfica, ya que permite convertir cualquier dato de tipo Num, a una cadena de caracteres (la función ‘show’ también permite llevar listas a una cadena), no es de orden superior debido a que no se reciben funciones como parámetro. A su vez es de orden superior dado que devuelve una función como resultado.

- **cadenaANum:** Permite llevar un String a un número usando la funcion ‘read’. Se clasificó esta funcion como polimórfica, ya que permite convertir cualquier dato de tipo String a un tipo de dato Num (cabe recalcar que debe ser un String que sea posible convertirlo a número), no es de orden superior debido a que no se reciben funciones como parámetro. A su vez es de orden superior dado que devuelve una función como resultado.

- **obtenerFactores:** Permite obtener los factores de un entero, cada uno de ellos almacenado en una lista. No es polimórfica ni de orden superior, solamente recibe una

lista de enteros y retorna una lista obtenida de evaluar la funcion filter (No retorna funciones, ni las recibe como parámetro, no es de orden superior).

- **aliquout:** Permite sumar los números de una lista y retornar el resultado. Solamente se clasificó como polimórfica dado que la lista puede ser de tipo Num. No se consideró de orden superior ya que las expresiones a entregar no corresponden a funciones sino evaluaciones de estas (se evaluó la funcion sum, con el resultado de obtenerFactores, todo esto con un parámetro que ya estaba definido en la funcion).

- **borrarElementos:** Permite borrar los 'n' primeros elementos de una lista, es polimórfica y de orden superior, esto debido a que puede aplicarse a una lista, con elementos de cualquier tipo. Es de orden superior ya que borrarElementos devuelve una funcion que está esperando una lista, es decir, es una funcion parcialmente aplicada.

- **cogerElementos:** Permite obtener los 'n' primeros elementos de una lista, es polimórfica y de orden superior, esto debido a que puede aplicarse a una lista, con elementos de cualquier tipo. Es de orden superior ya que cogerElementos devuelve una funcion que está esperando una lista, es decir, es una funcion parcialmente aplicada.

- **obtenerLista:** Esta funcion permite obtener una subcadena que inicia en un índice dado, y a partir de este índice, se toman los 'n' primeros números. Corresponde a una función polimórfica, dado que el parámetro que se define como 'numero' puede ser realmente cualquier tipo de lista, en este caso se usó para obtener una subcadena. No se considero de orden superior ya que solamente evalúa funciones y retorna su resultado, no las funciones.

- **obtenerPeriodo:** Corresponde a una función útil para obtener el periodo de ingreso de un estudiante. No puede ser polimórfica ya que toma como tipo de dato especifico un entero. No es de orden superior ya que solo devuelve evaluaciones de funciones o cadenas de caracteres (tampoco tiene como parámetro alguna función).

- **obtenerEscuela:** Corresponde a una función útil para obtener la escuela a la que pertenece un estudiante. No puede ser polimórfica ya que toma como tipo de dato especifico un entero. No es de orden superior ya que solo devuelve evaluaciones de funciones.

- **esPar:** Decide si un entero es par o no. No es polimórfica ni de orden superior, esto debido a que recibe un exclusivamente un entero y devuelve un string (no devuelve una funcion o toma como parametros alguna funcion, por lo que no es de orden superior).

- **obtenerNum:** Permite visualizar el numero ingresado, por ejemplo ‘num6’. Es polimórfica ya que el parámetro ‘num’ puede ser un número, una lista u otro tipo de dato. La función ‘show’ mapeará este dato en un string.

- **addZeros:** Añade ceros por la izquierda a una cadena en el caso de que su longitud sea inferior a tres. No es polimórfica ni de orden superior ya que recibe exclusivamente una cadena de caracteres, a su vez retorna una evaluación a una función o un string.

Enlace del repositorio y del video:
https://github.com/AmazingJuan/Practice-2
https://youtu.be/SnOHBZhMero