



ISBNchecker.hs

Descripción del Proyecto

Este proyecto consiste en la creación de un programa en Haskell para identificar las características de los códigos de inscripción de los estudiantes de pregrado en la Universidad EAFIT. El objetivo es analizar y desglosar los códigos de inscripción en sus componentes principales y proporcionar información relevante sobre cada uno.

Problema

Los empleados de la oficina de Admisiones y Registro de la Universidad EAFIT necesitan modificar la manera en que se genera el código de inscripción que se registra en la tarjeta de identificación del estudiante. Este código debe identificar:

- El período de admisión.
- El tipo de programa al que el estudiante ingresa (Administración, Humanidades o Ingeniería).
- Un número consecutivo que refleje el orden de admisión al programa.
- Indicar si el código es un valor numérico par o impar.

Proceso de Resolución del Ejercicio

1. Análisis del problema: Identificamos los componentes del código y las características que debían extraerse (período académico, categoría del programa, número consecutivo, paridad).
2. Diseño de la solución: Decidimos usar Haskell para implementar funciones que extraen y clasifican cada componente del código.
3. Implementación: Desarrollamos las funciones en Haskell, asegurándonos de seguir el esquema de clasificación de Nicomachus y las reglas de formato para el código.

Solución

La propuesta adoptada consiste en componer el código del estudiante con los siguientes elementos:

1. Los primeros tres dígitos corresponden al período de admisión (ej. 241 para 2024-1, 242 para 2024-2, etc.).
2. Los siguientes dos dígitos corresponden a la categoría del programa académico, utilizando el esquema de clasificación de Nicomachus (abundante, perfecto, deficiente).
3. Los últimos tres dígitos corresponden al número consecutivo de admisión, con valores de 001 a 999.
4. El código se analiza para determinar si es un valor numérico par o impar.

Implementación

El programa está implementado en Haskell y realiza las siguientes tareas:

1. Extrae los componentes del código de inscripción.
2. Determina el período académico basado en los primeros tres dígitos.
3. Clasifica la categoría del programa académico utilizando la suma alícuota y el esquema de clasificación de Nicomachus.
4. Obtiene el número consecutivo de admisión.

5. Determina si el código es par o impar.
6. Genera la salida esperada en el formato especificado.

Manifestación del Currying en la Solución Propuesta

El currying es una característica fundamental de Haskell que permite que las funciones tomen sus argumentos uno a uno, devolviendo una nueva función con cada argumento recibido. Aunque en la solución propuesta no se hace uso explícito del currying, todas las funciones en Haskell son, por defecto, funciones curriadas.

Por ejemplo, consideremos la función `isAbundant` :

```
isAbundant :: Int -> Bool
isAbundant n = aliquotSum n > n
```

Esta función toma un solo argumento `n` y devuelve un valor booleano. Implícitamente, podríamos considerar esta función como curriada, ya que si la reescribimos en una forma explícita de currying, sería:

```
isAbundant :: Int -> (Bool)
isAbundant = \n -> aliquotSum n > n
```

En esta forma, se puede ver que `isAbundant` es una función que toma un argumento `n` y devuelve otra función que compara la suma alícuota de `n` con `n`.

Clasificación de las Funciones

a. Polimórficas

En esta implementación específica, no se utilizaron funciones polimórficas. Las funciones polimórficas son aquellas que pueden operar sobre diferentes tipos de datos. En nuestro caso, todas las funciones trabajan exclusivamente con tipos de datos enteros (`Int`) y cadenas de caracteres (`String`). No hay necesidad de que las funciones operen sobre múltiples tipos de datos, lo cual es la razón por la cual no se utilizaron funciones polimórficas.

b. De Orden Superior

Las funciones de orden superior son aquellas que toman otras funciones como argumentos o devuelven funciones como resultados. En esta implementación, ninguna de las funciones desarrolladas cumple con este criterio. Esto se debe a que las operaciones necesarias para resolver el problema (como cálculos matemáticos y manipulación de cadenas) no requieren el uso de funciones de orden superior. La solución se basa en aplicar funciones directamente a los datos sin necesidad de pasar funciones como argumentos.

c. No Clasificadas

Las funciones desarrolladas en esta implementación se pueden clasificar en:

- **Funciones básicas de transformación y extracción:** `getPeriod`, `getConsecutive`, `getCategory`, `getParity`.
- **Funciones auxiliares de cálculo:** `isAbundant`, `isPerfect`, `isDeficient`, `aliquotSum`.

Cada una de estas funciones tiene un propósito claro y específico, y no se requiere que sean polimórficas o de orden superior. Su diseño se centra en realizar tareas específicas de transformación y cálculo de manera eficiente y directa.

Justificación de la No Utilización de Funciones Polimórficas y De Orden Superior

- **Funciones Polimórficas:** No se utilizaron debido a la naturaleza del problema, que se enfoca en trabajar con enteros específicos y cadenas de caracteres. La necesidad de manejar múltiples tipos de datos no está presente en este contexto.
- **Funciones de Orden Superior:** No se utilizaron porque las operaciones requeridas no involucran manipulación de funciones como datos. Las transformaciones y cálculos necesarios se pueden realizar directamente sobre los datos sin necesidad de pasar funciones como argumentos o devolver funciones como resultados.

Este enfoque simplifica la implementación y hace que el código sea más directo y fácil de seguir, manteniendo la solución alineada con los requisitos del problema.

Código

```

import Text.Printf (printf)
-- Función principal que toma un código y retorna sus características
studentCodeCharacteristics :: Int -> String
studentCodeCharacteristics code =
    let codeStr = printf "%08d" code
        period = getPeriod (take 3 codeStr)
        categoryCode = read (take 2 (drop 3 codeStr)) :: Int
    consecutive = getConsecutive (drop 5 codeStr)
        category = getCategory categoryCode
        parity = getParity code
    in period ++ " " ++ category ++ " " ++ consecutive ++ " "
++ parity
-- Función para obtener el período académico
getPeriod :: String -> String
getPeriod "241" = "2024-1"
getPeriod "242" = "2024-2"
getPeriod "251" = "2025-1"
getPeriod "252" = "2025-2"
getPeriod "261" = "2026-1"
getPeriod "262" = "2026-2"
getPeriod _ = "Unknown"
-- Función para obtener el consecutivo de admisión
getConsecutive :: String -> String
getConsecutive digits = "num"
++ show (read digits :: Int)
-- Función para determinar la categoría del programa
getCategory :: Int -> String
getCategory n
    | isAbundant n = "Administrative"    | isPerfect n = "Engineering"
    | isDeficient n = "Humanities"      | otherwise = "Unknown"
-- Función para verificar si un número es abundante
isAbundant :: Int -> Bool
isAbundant n = aliquotSum n > n
-- Función para verificar si un número es perfecto
isPerfect :: Int -> Bool
isPerfect n = aliquotSum n == n
-- Función para verificar si un número es deficiente
isDeficient :: Int -> Bool
isDeficient n = aliquotSum n < n
-- Función para calcular la suma alícuota
aliquotSum :: Int -> Int
aliquotSum n = sum [x | x <- [1..n-1], n `mod` x == 0]
-- Función para determinar si un número es par o impar
getParity :: Int -> String
getParity n
    | even n = "even"    | otherwise = "odd"
-- Ejemplos de uso
main :: IO ()

```

```
main = do      a <- readLn      putStrLn (studentCodeCharacteristics a)
```

Cómo usar

1. Clona este repositorio.
2. Compila el archivo `ISBNchecker.hs` usando GHC o cualquier otro compilador de Haskell.
3. Ejecuta el programa e ingresa un código de inscripción de 8 dígitos para obtener las características del código.

Licencia

Este proyecto está bajo la Licencia MIT.