Paradigmas de la Programación
Laboratorio 3: Programación Funcional

Laura Alonso Alemany Cristian Cardellino Ezequiel Orbe

En este laboratorio desarrollaremos un proyecto utilizando programación funcional, para lo cual reimplementaremos el corrector ortográfico (*spellchecker*) que hicimos en el Laboratorio 2 en uno de los lenguajes funcionales puros más conocidos: `Haskell`. Nuestro objetivo será poner a trabajar los conceptos más importantes de la programación funcional y poder comprender sus diferencias con respecto a la programación imperativa.

Características de la presentación

- El trabajo es en grupo: máximo: **dos integrantes**, mínimo: **dos integrantes**.
- Fecha de Entrega: Hasta las 23:59:59 del 30/04/2015.
- Formato de entrega:
  1. Empaquetar el directorio `spellchecker` en un archivo llamado `<dni>-lab-3.tar.gz`, donde `<dni>` es el DNI de alguno de los integrantes del grupo.
  2. Enviar el archivo `<dni>-lab-3.tar.gz` por mail a la dirección:

     `paradigmas@famaf.unc.edu.ar`

     El título del mail debe decir:`lab-3`. En el cuerpo del mail se deben indicar los integrantes (nombre completo y DNI).
  3. La nota asignada a los trabajos enviados fuera de término será computada de acuerdo a la siguiente Política de Entrega Tardía:
     - Hasta 1 día despues del deadline: -50% de la nota.
     - Hasta 2 días despues del deadline: -60% de la nota.
     - Hasta 3 días despues del deadline: -70% de la nota.
     - Hasta 4 días despues del deadline: -80% de la nota.
     - Hasta 5 días despues del deadline: -90% de la nota.
     - Más de 5 días despues del deadline: el trabajo se considera NO entregado y se debe recuperar.
  4. Los trabajos enviados que no cumplan con las condiciones de presentación no serán considerados.

Software Requerido

Para realizar este laboratorio necesitaremos el siguiente software:

- Compilador de Haskell (version 7.6.3 en adelante)
- Cabal

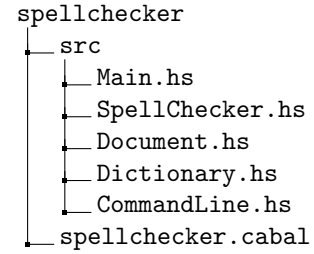
Se pueden instalar ambas herramientas instalando el Haskell Platform

Código Inicial

Para realizar este laboratorio, utilizaremos el código disponible en:

`http://cs.famaf.unc.edu.ar/materias/paradigmas/node/187`

Al extraer el contenido del archivo, tenemos la siguiente estructura de directorios:



1 Especificaciones del Corrector Ortográfico

El corrector ortográfico (spellchecker) que desarrollaremos deberá utilizar un diccionario de palabras conocidas, el cual se cargará en memoria desde un archivo cuando el programa se inicie, y un diccionario de palabras ignoradas, el cual inicialmente estará vacío.

Dado un documento de entrada, el spellchecker copiará a un documento de salida todas las palabras y los signos de puntuación, consultando interactivamente al usuario sobre cada palabra desconocida.

Dada una palabra desconocida, el usuario podrá aceptarla, ignorarla o reemplazarla.

Si el usuario acepta la palabra, la misma se agregará al diccionario, si el usuario ignora la palabra, esta ocurrencia y las subsiguientes se ignorarán. Finalmente, si el usuario decide reemplazar la palabra, el sistema permitirá que el usuario ingrese una nueva palabra que reemplazará a la palabra desconocida en el documento de salida.

Una vez que todo el documento de entrada haya sido procesado, el programa guardará el diccionario en el mismo archivo desde el cual lo cargó al inicio y dejará disponible un nuevo archivo que contenga el documento de salida.

1.1 Funcionalidades deseadas

1. Leer un diccionario de palabras conocidas desde un archivo.
2. Tener en memoria un diccionario de palabras ignoradas.
3. Leer un documento de entrada, palabra por palabra.
4. No proponer acciones para palabras conocidas.
5. Proponer tres acciones posibles para palabras desconocidas: `aceptar`, `ignorar`, `reemplazar`.
6. En caso de aceptar una palabra desconocida: incorporarla al diccionario de palabras conocidas y que las subsiguientes ocurrencias de la palabra sean tratadas como conocidas.
7. En caso de ignorar una palabra desconocida: incorporarla al diccionario de palabras ignoradas e ignorar la palabra y sus subsiguientes ocurrencias.
8. En caso de reemplazar una palabra desconocida: en el documento de salida se escribirá la palabra por la que se reemplaza, en lugar de la original.
9. Actualizar el diccionario de palabras conocidas incluyendo las palabras aceptadas.
10. Escribir un documento de salida con las palabras originales o sus modificaciones, según las acciones realizadas por el usuario.

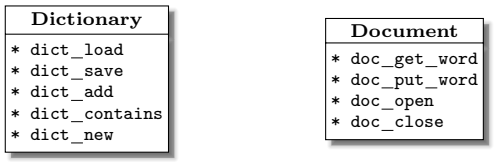


Figure 1: Interface de los TADS `Dictionary` y `Document`

2 Consignas

- Implementar el TAD Dictionary (`Dictionary.hs`). El mismo debe ser desempaquetado, seguro y debe respetar la interface especificada en la Figura 1. Las operaciones especificadas en esta figura se encuentran detalladas en el archivo `Dictionary.hs`.
- Implementar el TAD Document (`Document.hs`). El mismo debe ser desempaquetado, seguro y debe respetar la interface especificada en la Figura 1. Las operaciones especificadas en esta figura se encuentran detalladas en el archivo `Document.hs`.
- Implementar el módulo SpellChecker (`SpellChecker.hs`).
- No te olvides de verificar que la implementación funciona correctamente, es decir, que tiene las funcionalidades especificadas en la Sección 1, mediante ejecuciones de ejemplo para cada funcionalidad. Para compilar con `cabal`: `cabal build` (la primera vez que compiles necesitarás ejecutar `cabal configure`).

2.1 Aclaraciones

- Se deben respetar los tipos de las funciones especificadas en el código.
- Se pueden utilizar funciones auxiliares.
- Los comentarios de las funciones, en el código entregado, dan más detalles sobre lo que debe hacer cada una de ellas.
- Es fundamental leer la documentación de Haskell. Usen Hoogle.
- `cabal` es el sistema de gestión de paquetes de `Haskell`. En nuestro caso lo utilizamos como el `makefile` para los proyectos en `C`.