Paradigmas de la Programación
Laboratorio 2: Programación Imperativa

Laura Alonso Alemany Ezequiel Orbe

En este laboratorio desarrollaremos un proyecto utilizando el paradigma de programación imperativo, para lo cual desarrollaremos un corrector ortográfico (*spellchecker*) en el lenguaje imperativo más conocido: `C`. Nuestro objetivo será poner a trabajar los conceptos más importantes del paradigma imperativo y comprender cómo afectan a la arquitectura de un programa.

Características de la presentación

- El trabajo es en grupo: máximo: **dos integrantes**, mínimo: **dos integrantes**.
- Fecha de Entrega: Hasta las 23:59:59 del 16/04/2015.
- Formato de entrega:
  1. Empaquetar el directorio `spellchecker` en un archivo llamado `<dni>-lab-2.tar.gz`, donde `<dni>` es el DNI de alguno de los integrantes del grupo.
  2. Enviar el archivo `<dni>-lab-2.tar.gz` por mail a la dirección:

     `paradigmas@famaf.unc.edu.ar`

     El título del mail debe decir:`lab-2`.
  3. Los trabajos enviados fuera de término o que no cumplan con las condiciones de presentación no serán considerados.

Software Requerido

Para realizar este laboratorio necesitaremos el siguiente software:

- Compilador de C

Código Inicial

Para realizar este laboratorio, utilizaremos el código disponible en:

`http://cs.famaf.unc.edu.ar/materias/paradigmas/sites/default/files/labs/lab-2.tgz`

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

```
spellchecker
├── tightly-coupled
│   └── src
└── loosely-coupled
    └── src
```

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.

Funcionalidades deseadas:

1. Leer un diccionario de palabras conocidas.
2. Leer un diccionario de palabras ignoradas.
3. Leer un documento de entrada, palabra por palabra y separando signos de puntuación.
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 y que las subsiguientes ocurrencias de la palabra sean tratadas como conocidas.
7. En caso de ignorar una palabra desconocida: 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.

2 Arquitectura de un programa imperativo

Los conceptos clave (variables y procedimientos) del paradigma imperativo afectan la arquitectura así como la codificación de los programas imperativos.

Por arquitectura de un programa se entiende la forma en la cual un programa se descompone en componentes, es decir, en diferentes unidades de programa, junto con las relaciones existentes entre estos componentes.

Desde el punto de vista de la ingeniería de software la arquitectura de un programa es muy importante ya que afecta directamente a los costos de implementación y al costo de mantenimiento.

Una medida de calidad importante es el acoplamiento (coupling) que existe entre los distintos componentes, y que indica cuán sensibles son los componentes a los cambios realizados en otros componentes.

Un programa está fuertemente acoplado (tightly coupled) si las modificaciones en uno de sus componentes fuerza modificaciones importantes en los demás componentes. Un programa está

levemente acoplado (loosely coupled) si las modificaciones en uno de sus componentes fuerzan, a lo sumo, modificaciones menores en los demás componentes.

Idealmente queremos que nuestros programas esten levemente acoplados, debido a que un programa desarrollado de esa forma es más fácil de mantener en el tiempo ya que cada componente se puede modificar independientemente.

Un programa imperativo tradicional consiste de procedimientos y variables globales, donde los procedimientos se llaman unos a otros y acceden a las variables globales.

Que los procedimientos se llamen unos a otros no es un problema, ya que una modificación en la implementación de uno de ellos no afecta la implementación de los demás procedimientos. Sin embargo, el acceso a variables globales crea un fuerte acoplamiento, ya que cualquier modificación en la representación de una variable global forzará la reimplementación de los procedimientos que la acceden. Además, estos procedimientos son sensibles a la forma en que una variable es inicializada, inspeccionada y modificada por los otros procedimientos.

Para entender las implicaciones de diseñar una arquitectura fuertemente acoplada o una arquitectura levemente acoplada, desarrollaremos 2 versiones de nuestro spellchecker.

3 Spellchecker: Arquitectura Fuertemente Acoplada

La primera versión del spellchecker será una versión fuertemente acoplada.

La Figura 1 presenta la arquitectura del spellchecker. En la misma podemos observar lo siguiente:

- Existe una variable global `dict_main` que contiene el diccionario que ha sido cargado desde un archivo, y que luego será guardado en el mismo archivo.
- Existe una variable global `dict_ignored` que contiene el diccionario de palabras ignoradas por el usuario.
- Las variables globales `doc_in` y `doc_out` refieren a los archivos de entrada y salida respectivamente.
- Los procedimientos de bajo nivel, `dict_load,dict_save, dict_add, ignored_add, is_known` y `get_word,put_word` operan sobre las variables globales.
- Los procedimientos de alto nivel, `consult_user, process_document` y `main` se encargan de llamar a los procedimientos de bajo nivel.

3.1 Consignas

1. Completá la implementación del spellchecker en `tightly-coupled/src/spellchecker.c` para que tenga las funcionalidades especificadas en la Sección 1. En los comentarios del código hay requerimientos adicionales.

   IMPORTANTE: No se puede modificar la arquitectura del programa.

2. 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.

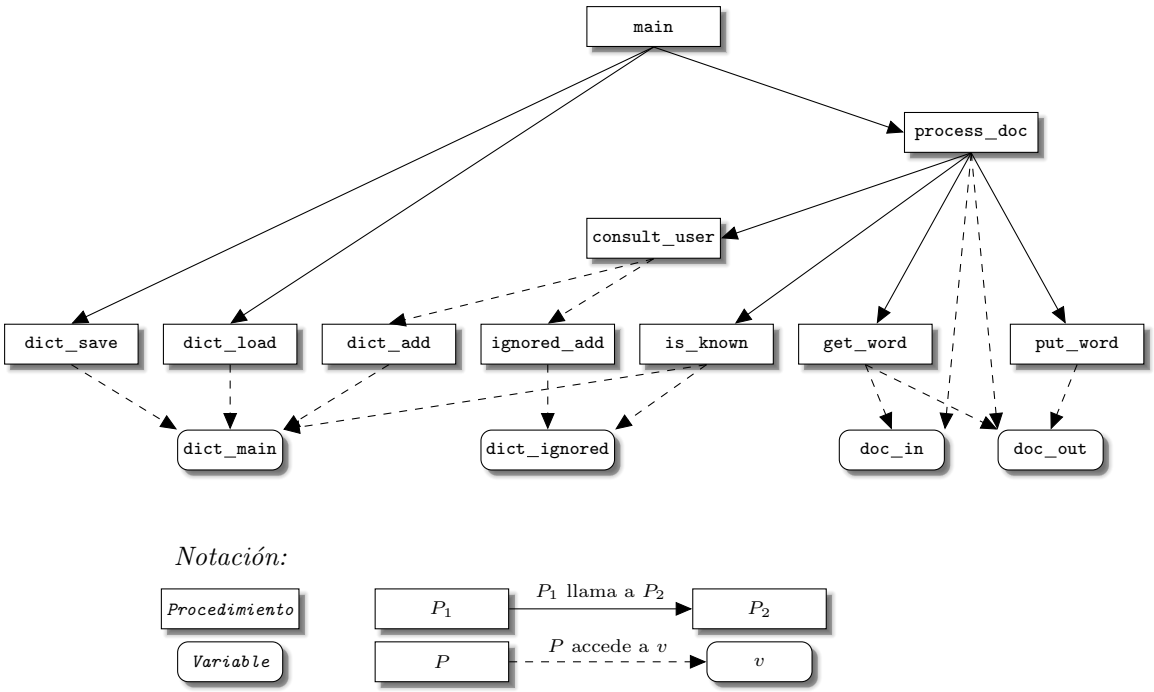

Figure 1: Spellchecker: arquitectura altamente acoplada

4 Spellchecker: Arquitectura Levemente Acoplada

La Figura 2 presenta una arquitectura levemente acoplada para el spellchecker. En la misma podemos observar lo siguiente:

- Se han definido dos Tipos Abstractos de Datos (TADs): `Dictionary` y `Document`, los cuales representan un diccionario y un documento respectivamente.
- Los procedimientos de bajo nivel, `dict_load, dict_save, dict_add, ignored_add` y `get_word, put_word`, ahora son parte de las operaciones definidas por los TADs.
- Las que en la arquitectura acoplada eran las variables globales `doc_in` y `doc_out`, ahora se fusionaron en una sola variable global `doc`.

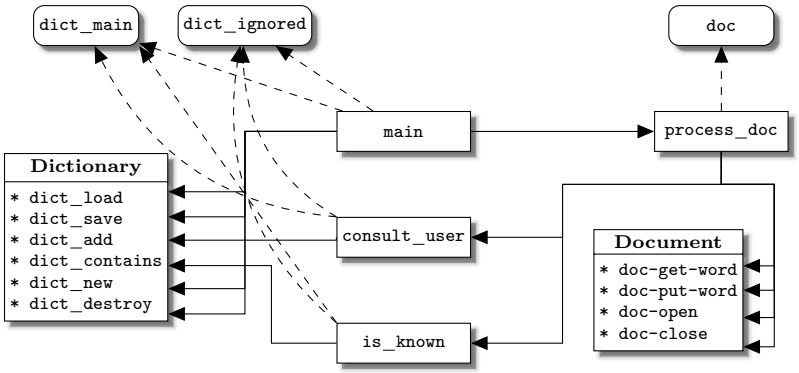

Figure 2: Spellchecker: arquitectura levemente acoplada

- Las variables globales `dict_main`, `dict_ignored`, y `doc` representan estructuras de datos sobre las cuales pueden operar las operaciones definidas por los TADs.
- Los procedimientos de alto nivel, `consult_user, process_document` y `main` se encargan de invocar las operaciones definidas por cada TAD sobre las variables correspondientes.

4.1 Consignas

1. Completá la implementación del spellchecker en `loosely-coupled/src/spellchecker.c`, la cual sigue la arquitectura levemente acoplada de la Figura 2. Tené en cuenta lo siguiente:

   (a) Los TADs `Dictionary` y `Document` deben ser implementados de forma tal que sean desempaquetados y seguros (hint: Utilizar punteros opacos).

   IMPORTANTE: No se puede modificar la arquitectura del programa.

2. 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.