Tarea #1
Lenguajes de Programación INF-253
República Democrática Esclavista de Elepé

Sven von Brand
svbrand@inf.utfsm.cl

Rodolfo Castillo
rodolfo.castillo.13@sansano.usm.cll

Marcelo Jara
marcelo.jara.13@sansano.usm.cl

Segundo semestre académico, 2015

1 Objetivos

- Que el alumno refuerce los conceptos aprendidos en *estructuras de datos*, explotando esta vez el potencial del lenguaje de programación C.
- Que aplique nociones avanzadas de punteros, incluyendo asignaciones por referencia, punteros a función, etc.
- Que implemente en la práctica una forma básica de *parse tree.*

2 Introducción

El país democrático de Elepé ha solicitado a sus ciudadanos participar de un desarrollo de su nuevo sistema de predicción de resultados de licitaciones.
Los dignatarios de Elepé no confían en el software libre y la nación es pobre, por lo que no tienen dinero para despilfarrar en software privativo.
Sin embargo en Elepé aún no se abole la esclavitud, así que se le ordenó a los esclavos informáticos estatales (E.I.E.) que realicen prototipos de DBMS (DataBase Management System) sencillos en agrupaciones de hasta 2 ingrantes. De esta manera el estado no paga por SW adicional y puede registrar la participación de los ciudadanos.

3 La Tarea

3.1 Descripción

En esta tarea deberán ser capaces de asignar secuencias de strings (llámense comandos) a alguna acción predeterminada. Para esto utilizarán una estructura de árbol donde almacenarán los caminos que siguen los distintos comandos, para desembocar en una función. De esta manera tendrán que recorrerlo y compararlo con cada elemento de un input para finalmente ejecutar la acción que corresponda o en su defecto no ejecutar nada si no existe alguna ocurrencia que coincida. A continuación se muestra como se vería gráficamente un árbol como el descrito.

3.2 Ejemplo demostrativo

El siguiente es un ejemplo sencillo, empezando con el input y output esperado:

```
4
>> 1 say my name
>> 2 say hello to my little friend
>> 3 say hello to my purple friend
>> 0 hasta la vista baby

> say my name carmack.txt
>>> Your name is John Carmack
> say hello to my little friend bjarne.txt
>>> Hello to you, Bjarne Stroustrup
> hasta la vista baby chao.txt
>>> The program has been terminated
```

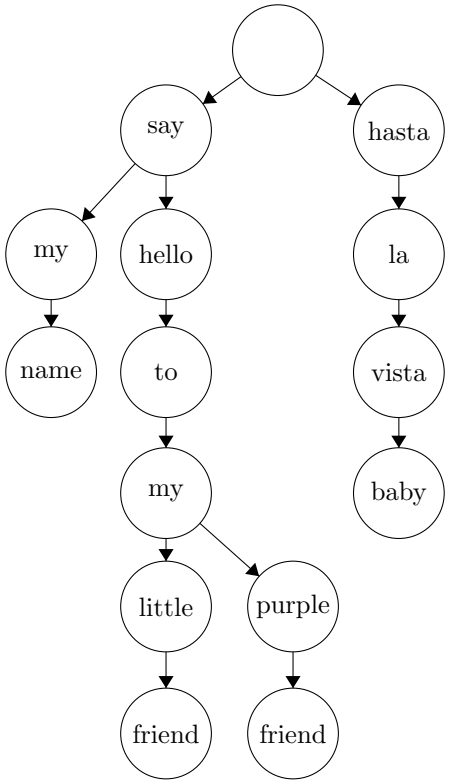
Del ejemplo anterior:

- Se lee la cantidad de instrucciones que cargará el programa en la primera línea.
- Desde la línea 2 a la 5 se leen las instrucciones que se cargaran en el programa, de la forma: `<id_callback> <tokens...>`, donde `id_callback` es el id del callback a llamar, los cuales serán especificados en la siguiente sección.
- Las siguientes líneas serán un espacio abierto para ingresar comandos que el programa deberá ejecutar basados en las instrucciones cargadas anteriormente. Estos comandos serán de la forma: `<tokens...> <input_file>`, es decir, la última palabra es el nombre del *archivo de input*. Con estos comandos usted deberá recorrer el árbol de instrucciones y ejecutar el callback con el archivo de input correspondiente.

**Los callback utilizados en el ejemplo no tienen relación con los exigidos.*

El archivo de input aludido anteriormente deberá ser abierto en cada una de las funciones sirviendo de argumentos. Además existe el nombre reservado ***noargs*** *que significa que no se debe abrir ningún archivo*

A partir de esto se genera el siguiente parse tree:



3.3 Callbacks

Se piden crear los siguientes callbacks:

** A excepción del callback **crear**, si el archivo no existe, entregue un error y continue con el resto de líneas.*

- **Terminar programa (id: 0)** Terminar el programa, entregando el mensaje almacenado en el archivo de input. Si se entrega ***noargs*** como argumento: se termina la ejecución del programa sin entregar mensaje.
- **Insertar (id: 1)** Insertar todos los datos en los archivos correspondientes en la línea indicada. El formato de cada línea del archivo es: `<archivo> <linea> <dato_1> <dato_2> <dato_n>`. En caso de que la línea no exista puede rechazarla y continuar con la siguiente y en caso que la línea tenga un valor menor a 0 (cero) se debe agregar al final. En el caso que el archivo no exista debe notificarlo. Debe entregar un porcentaje de la cantidad de líneas que se pudieron agregar en total.

  Ejemplo de archivo de input:
  ```
  alumnos 3 juan pizarro 21 m
  alumnos 8 almendra diaz 20 f
  ```

  Ejemplo de output:
  ```
  Se han insertado 1 de 2 registros (50%) satisfactoriamente.
  ```

- **Eliminar por línea (id: 2)** Por cada línea del archivo de input debe eliminar el dato que se pide en cierto archivo. El dato será el indicado por el número de línea. El formato de cada línea es: `<archivo> <número_linea>`. Debe entregar los datos de cada línea eliminada. En caso de que no exista la línea debe entregar un mensaje de error y continuar con la siguiente línea. Si se especifica un número menor a 0 (cero) como línea entonces debe eliminar la última línea.

  Ejemplo de archivo de input:
  ```
  alumnos 3
  alumnos 8
  ```

  Ejemplo de output:
  ```
  juan pizarro 21 m
  Linea 8 no existe en archivo alumnos.
  ```

- **Eliminar por coindicencia (id: 3)** Por cada línea del archivo de input debe eliminar cierta línea de cierto archivo dado el índice del dato el cual comparar y el keyword a comparar. El formato de cada línea será de la forma: `<archivo> <índice_dato> <keyword>`. El número del dato parte contando desde 0 y el keyword es siempre una sola palabra (sin espacios).

Si el índice no existe debe entregar un error y seguir con el resto.

Ejemplo de archivo de input:
```
alumnos 2 21
jugadores 4 futbol
```

En el ejemplo se pide eliminar todos los datos del archivo alumnos cuyo dato de índice es el 2 (por ejemplo la edad) es 21. Luego se pide eliminar todos los datos del archivo jugadores donde el dato índice 4 sea futbol.

- **Mostrar por línea (id: 4)** Por cada línea del archivo de input debe mostrar los datos en cierta línea de cierto archivo. El formato de cada línea será de la forma: `<archivo> <línea>`. Si la línea no existe entregue un error y siga con el resto de líneas.

  Ejemplo de archivo de input:
  ```
  alumnos 2
  ```

- **Crear archivo (id: 5)** Debe poder crear todos los archivos separados por saltos de línea. Si el archivo ya existía, debe eliminar su contenido.

  Ejemplo de archivo de input:
  ```
  alumnos
  atletas
  ```

- **Eliminar archivo (id: 6)** Debe eliminar todos los archivos específicados en el archivo de input, separados por saltos de línea. Si el archivo no existe debe indicarlo.

  Ejemplo de archivo de input:
  ```
  alumnos
  atletas
  ```

- **Truncar archivo (id: 7)** Debe limpiar el contenido de todos los archivos indicados separados por saltos de línea.

  Ejemplo de archivo de input:
  ```
  alumnos
  jugadores
  pintores
  ```

Todos los archivos deben ser guardados en una carpeta especial llamada **db**, la cual puede asumir que siempre se encuentra creada (no debe crearla si no existe).

Puede definir una extensión especial para los archivos, sin embargo, no es requerimiento.

Además para optar al bonus debe implementar el siguiente callback:

- **Mostrar datos ordenados (id: 9)** Debe poder seleccionar los datos y ordenarlos ascendente o descendentemente dado el índice del dato. El archivo sólo contendrá una línea la cual indicará el archivo, el índice del dato y el tipo de orden (asc|desc). Debe mostrar todos los datos ordenados.

  Ejemplo de archivo de input:
  `alumnos 3 desc`

  Por lo tanto, si el archivo alumnos tiene datos del tipo: `<nombre> <apellido paterno> <apellido materno> <edad>`, entonces se ordenarán por edad.

A continuación, para elimiar incertidumbre se presenta un ejemplo más elaborado con varios aspectos de la tarea (omita los símbolos `#` ya que serán utilizados a modo de comentario para explicar el flujo y no son parte de la tarea. **DISCLAIMER:** por motivo de protección de información de la república democrática esclavista de Elepé los datos presentados a continuación no tienen correlación alguna con la realidad):

```
6
>> 1 it does not matter the length of the command it should work
    anyways # Con esto se deja a entender que los comandos pueden ser
    de largo ilimitado. Este comando inserta datos (id 1).
>> 1 we can perfectly have more than one command bound to the same
    action # Notamos que no hay problemas en definir mas de un
    comando para el mismo procedimiento.
>> 3 DELETE FROM FILE WHERE INDEX=KEYWORD
>> 0 we can perfectly do whatever we want # Inicio se parece a un
    comando anterior
>> 4 it does not matter # Puede ser un comando que este contenido en
    uno mas grande
>> 5 howdoyouturnthison

> we can perfectly have more than one command bound to the same action
    populate.cmds
>>> El archivo db/LP.db no existe
>>> El archivo db/LP.db no existe
>>> El archivo db/ramos.txt no existe
>>> El archivo db/LP.db no existe
>>> El archivo db/ramos no existe
> howdoyouturnthison create.cmds
>>> Archivo creado: db/LP.db
>>> Archivo creado: db/ramos
> it does not matter the length of the command it should work
    populate.cmds
>>> El archivo db/ramos.txt no existe
>>> Se han insertado 4 de 5 registros (80\%) satisfactoriamente.
```

```
> it does not matter show.cmds
>>> Marcelo Ayudante
> DELETE FROM FILE WHERE INDEX=KEYWORD delete.cmds
>>> Se elimino [Marcelo|Ayudante] de db/LP.db satisfactoriamente.
>>> No existe 1: 40 en db/ramos
> it does not matter show.cmds
>>> Rodolfo Ayudante
> we can perfectly do whatever we want noargs
```

Considerar los archivos de input:

populate.cmds

```
LP.db -1 Sven Profesor
LP.db -1 Marcelo Ayudante
ramos.txt 2 MAT021 13
LP.db -1 Rodolfo Ayudante
ramos 0 INF253 100
```

create.cmds

```
LP.db
ramos
```

show.cmds

```
LP.db 1
```

delete.cmds

```
LP.db 0 Marcelo
ramos 1 40
```

Importante: El input y output de su programa deben corresponder a los formatos utilizados en los ejemplos, donde:

- `>>>`: Indica una respuesta de su programa (salida).
- `>>`: Indica al usuario que ingrese una instrucción en la primera etapa del programa.
- `>`: Indica al usuario que ingrese un comando a ejecutar.

Los mensajes de errores puede escribirlos como desee, pero siempre debe entregar información al usuario sobre su error. Es crucial este *feedback* para el usuario.

4 Consideraciones adicionales

- Usted puede realizar supuestos respecto a su tarea en el archivo README, serán considerados.
- No se exige una estructura determinada para la entrega de la tarea. Ustedes deben escogerla y explicarla en el archivo README. (Por ejemplo: Escogimos poner todo en un archivo porque nos pareció más fácil)

  Consejo 1: No siga el ejemplo.
  Consejo 2: Hágale caso al consejo 1.
  Consejo 3: Hágale caso al consejo 2.
  Condejo n: Hágale caso al consejo $n-1, n \in [4, \infty[$
- La limpieza de la memoria pedida en el heap es de suma importancia. Dependerá de su solución el como puedan abordar limpiar la memoria una vez que se llame al comando de salida. Como ayuda, una de las maneras posibles es mediante la función `at_exit` (investigue su uso).
- Como ayuda, se entregará la definición de un nodo del arbol y de una lista, las cuales son suficientes para la implementación por lo que no debería agregar datos. Además, se entregará la función `split`, la cual podrá utilizar libremente en su tarea. Será deber de usted aprender a usarla. Se entregarán ambos elementos en un tar.gz.
- Los callback sólo reciben el nombre del archivo por parámetro.
- No puede usar la función `atoi`.
- Debe entregar un error cuando el comando no exista.
- Debe verificar la existencia del archivo de input, en caso de que no exista, debe entregar un error.
- Solo se usarán caracteres ASCII.
- La tarea debe compilar en el ambiente del Laboratorio de Computación (LabComp).
- Los comandos siempre tendrán al menos dos palabras.

5 Evaluación del código

- Uso correcto del lenguaje (punteros y otros).
- Uso general del paradigma imperativo.
- Entrada/Salida del programa.
- Limpieza correcta (muy importante).

- Uso correcto de las estructuras de datos exigidas.
- Debe escribir un código ordenado y consistente en sus convenciones (investigue al respecto si es necesario).

6 Bonificaciones

- Implementar el callback número 9 (10 puntos). Indique en el archivo README si la implementó.

7 Sobre la entrega

- La tarea debe entregarse antes de las 23:55 hrs. del día **13 de noviembre del 2015**.
- Para consultas sobre el reglamento de tareas y/o el desarrollo de este enunciado, por favor dirigirse a la sección correspondiente en la plataforma Moodle.