Algoritmos y Programación II (75.41)
Trabajo Práctico Nº2

16 de septiembre de 2013

1. Introducción

Dados los problemas administrativos que está teniendo la actual organización del mundial de Brasil 2014, el mandamás de la FIFA decidió armar un software que le permita poner la casa en orden. Para esto, decidió asesorarse con su amigo íntimo Don Julio, quien le comentó que sponsorea un equipo de la FIUBA que 'programa mejor de lo que juega'.

Es por eso que la FIFA nos encargó la creación de un software (de ahora en más 'Messi +10') que permita organizar el fixture del mundial, pero con algunas limitaciones. Dado que lo van a usar mucho tiempo y la generación actual ya no es tan paciente, es requisito usar estructuras que sean de rápido acceso.

Como va a funcionar de backend para otro sistema, se necesita que el mismo presente una interfaz de comandos automatizable, que tome su entrada desde la entrada estándar. Como simplificación, se manejará la parte de eliminación directa del torneo y no la fase de grupos.

El fixture viene cargado en un archivo de texto, en el cual se dictaminan los partidos de inicio. Con los resultados del mismo (que serán cargados por comando) es que se llena el fixture. El sistema maneja un código para el manejo de partidos, en el cual a cada fase se le da el nombre de una potencia de 2^h, siendo H el número de instancia, contando como $h = 0$‘ la final. Por ejemplo, cuartos de final es 4. Cada rama de la instancia será nomenclada con una letra para especificar la llave a la que hace referencia. Por ejemplo, el primer partido de cuartos será 4a.

2. Consigna

Implementar en C el sistema de gestión de partidos del mundial, que debe proveer las siguientes operaciones:

- Carga inicial de archivo:
  - Armado de Fixture
  - Carga de equipos
  - Carga de jugadores
- Operaciones de administración:
  - Agregar resultado
- Servicios para consultas:
  - Listar goleador
  - Listar equipos
  - Listar jugadores según filtros
  - Consultar goles de un jugador
  - Consultar resultados por código

2.1. Protocolo

Este programa formará parte de un sistema automatizado. En lugar de presentar un menú con opciones en la pantalla, se esperará que el programa respete un *protocolo de comunicación*, en el cual se reciben comandos por la entrada estándar (`stdin`). Cada comando recibido debe ser procesado y su resultado debe ser escrito en la salida estándar (`stdout`).

Cada línea de la entrada corresponde a un comando, y todos los comandos respetan el mismo formato. El nombre del comando está separado de los parámetros (si existen) por un espacio, y los parámetros están separados por comas:

```
comando parametro1,parametro2,parametro3,...
```

2.2. Comandos de administración

2.2.1. Comando: `agregar_resultado`

Formato: `agregar_resultado idr,gloc,gvis[,dors. gol loc][,dors. gol vis]`

Descripción: Agrega un nuevo resultado al fixture.

Parámetros:

`idr`: Identificador del partido.

`gloc`: Goles del local.

`gvis`: Goles del visitante.

`nloc`: Dorsal de los jugadores que anotaron (local).

`nloc`: Dorsal de los jugadores que anotaron (visitante).

Salida: `OK` en caso de no producirse error. Mensaje de error apropiado en caso contrario (ver sección 2.4).

Requerimiento: Sea al menos $\mathcal{O}(\log(n) + k)$ siendo n la cantidad de partidos totales del fixture y k la cantidad de jugadores.

Ejemplo:

```
agregar_resultado 8a,3,1,11,9,11,14
OK
```

2.3. Servicios para consultas

2.3.1. Comando: `listar_jugadores`

Formato: `listar_jugadores (dorsal/nombre) equipo`

Descripción: Muestra una lista de los jugadores de un equipo, dependiendo del filtro seleccionado.

Parámetros:

`equipo`: Nombre del equipo a mostrar.

`dorsal/nombre`: Filtro a aplicar un ordenamiento de orden ascendente. Se debe poner uno o el otro.

Requerimiento: Sea al menos $\mathcal{O}(k)$ siendo k la cantidad de jugadores totales del equipo.

Salida: Un jugador por línea, con el siguiente formato:

```
<nombre>,<dorsal>: Goles: <goles convertidos>
```

Ejemplo:

```
listar_jugadores dorsal Argentina
Romero,1: Goles: 0
Fernandez,2: Goles: 0
Rojo,3: Goles: 0
Zabaleta,4: Goles: 0
Mascherano,5: Goles: 0
Garay,6: Goles: 0
Di Maria,7: Goles: 1
Gago,8: Goles: 0
Higuain,9: Goles: 4
Messi,10: Goles: 5
Aguero,11: Goles: 2
```

2.3.2. Comando: `listar_goleador`

Formato: `listar_goleador`

Descripción: Muestra una lista de los m goleadores del fixture.

Requerimiento: Debe tener un tiempo no menor a $\mathcal{O}(1)$.

Salida: Un jugador, con el siguiente formato:

```
<nombre>: <equipo> Goles: <goles>
```

Ejemplo:

```
listar_goleador
Chitalu: Zenegal Goles: 103
```

2.3.3. Comando: `goles_jugador`

Formato: `goles_jugador <nombre>`

Descripción: Muestra los goles de un jugador determinado.

Parámetros:

`nombre`: nombre del jugador a buscar.

Requerimiento: Debe tener un tiempo no menor a $\mathcal{O}(1)$.

Salida: Un jugado, con el siguiente formato:

```
<nombre>,<dorsal>: <equipo> Goles: <goles>
```

Ejemplo:

```
goles_jugador Maradona
Maradona,10: Argentina Goles:34
```

2.3.4. Comando: `mostrar_resultado`

Formato: `mostrar_resultado idr`

Descripción: Muestra el resultado del partido dado.

Parámetros:

`idr`: id del partido a mostrar.

Requerimiento: Sea al menos $\mathcal{O}(\log(n))$ siendo n la cantidad de partidos totales del fixture.

Salida: Un partido, con el siguiente formato:

```
resultado: <equipo>:<goles> vs <equipo>:<goles>
```

Ejemplo:

```
mostrar_resultado 2a
resultado: Argentina:2 vs Brasil:1
```

2.4. Mensajes de error

El sistema debe imprimir los siguientes mensajes de error, en cada uno de los casos de error descriptos a continuación:

Id repetido:

```
Error: el resultado con id <idr> ya existe
```

Resultado inexistente:

```
Error: el resultado con id <idr> no existe
```

Equipo ya inscripto en fixture:

```
Error: el equipo <nombre equipo> ya esta inscripto en ↩
    ↪el fixture
```

Equipo no inscripto en fixture:

```
Error: el equipo <nombre equipo> no esta inscripto en ↩
    ↪el fixture
```

Jugador no inscripto:

```
Error: el jugador <nombre jugador> no esta inscripto en ↩
    ↪el fixture
```

3. Carga de archivo

3.1. formato

```
nombre equipo 1
nombre jugador 1
nombre jugador 2
nombre jugador 3
...
nombre jugador 23
nombre equipo 2
....
nombre equipo N
nombre jugador 1
nombre jugador 2
nombre jugador 3
...
nombre jugador 23
```

Donde N = 2^k, siendo k cualquier numero natural. Los equipos se van juntando de a pares según vienen repartidos en el archivo.

3.2. Parseo

Los equipos están dados en el orden que llenarían las hojas del fixture. El parseo posee la limitación de que se debe parsear como viene, sin poder utilizar algoritmos de ordenamiento sobre los datos leídos.

4. Pruebas

Junto con la especificación se provee de **pruebas automáticas para el programa completo**. Estas pruebas serán de utilidad para revisar que el programa cumpla con la especificación del protocolo de entrada/salida.

Para la aprobación del Trabajo Práctico es requisito que el programa implementado pase todas las pruebas.

4.1. Ejecución de las pruebas

Una vez descomprimido el archivo zip con las pruebas[1], se debe efectuar los siguientes pasos para correr las pruebas:

1. Compilar el programa (supongamos que se llama `tp2`)
2. Ejecutar[2]:

```
$ bash pruebas/correr-pruebas.sh ./tp2
```

Cada una de las pruebas está especificada en un archivo con extensión `.test` dentro de la carpeta de pruebas. Por ejemplo:

```
pruebas/
 \- correr-pruebas.sh
 \- prueba1.test
 \- prueba2.test
```

El script `correr-pruebas.sh` ejecutará el programa una vez por cada prueba, pasándole en la entrada estándar los comandos especificados en la prueba. Luego verificará que la salida estándar del programa sea exactamente igual a la esperada según el protocolo.

5. Criterios de aprobación

A continuación describimos criterios y lineamientos que deben respetarse en el desarrollo del trabajo.

[1] Puede descomprimirse en cualquier lugar; para el ejemplo suponemos que se guardó en la misma carpeta que el TP. Es decir, el ejecutable `tp1` quedaría al mismo nivel que la carpeta `pruebas` (que contiene el archivo `correr-pruebas.sh`).

[2] Para correr las pruebas es necesario disponer de un entorno con línea de comandos Bash y herramientas GNU. En Linux seguramente no sea necesario instalar nada. Existen varias implementaciones para Windows; por ejemplo MSYS o Cygwin. Para Mac OSX existe el paquete `coreutils`

5.1. Utilización de estructuras de datos

Para la realización de este trabajo es necesario utilizar las estructuras de datos vistas en clase (arbol binario de búsqueda, hash, heap), además de crear las estructuras adicionales que se consideren necesarias.

Todas las estructuras deben estar implementadas de la forma más genérica posible y correctamente documentadas.

5.2. Programa

El programa debe cumplir los siguientes requerimientos:

- Debe estar adecuadamente estructurado y modularizado, utilizando funciones definidas de la forma más genérica posible, sin caer en lo trivial.
- El código debe ser claro y legible.
- El código debe estar comentado y las funciones definidas, adecuadamente documentadas.
- El programa debe compilar sin advertencias ni mensajes de error[3], debe correr sin pérdidas de memoria, uso de valores sin inicializar, o errores en general. Es decir que, el programa debe correr en valgrind sin errores.
- Además, claro, debe satisfacer la especificación de la consigna y pasar todas las pruebas automáticas.

5.3. Informe

El informe deberá consistir de las siguientes partes:

- **Carátula** con la información del alumno/a y el ayudante asignado (en caso de saberlo de antemano).
- **Análisis y diseño**: Describir la solución elegida para resolver cada problema propuesto, y cómo se lleva a cabo. En particular, mencionar cómo es el flujo del programa, qué algoritmos y estructuras de datos se utilizan, y cuál es el orden de ejecución en tiempo y espacio de cada operación.
- **Implementación**: Incluir aquí *todo* el código fuente utilizado (en formato `monoespaciado`, para facilitar su lectura).

[3] `-Wall -Werror -pedantic -std=c99`

- También *opcionalmente,* toda explicación adicional que consideren necesaria, referencias utilizadas, dificultades encontradas, cambios o mejoras que se podrían hacer a futuro y conclusiones.

El informe debe estar lo más completo posible, con presentación y formato adecuados. Por ejemplo, este enunciado cumple con los requerimientos de un informe bien presentado.

6. Entrega

El trabajo consiste en:

- El informe impreso.
- El informe digital, en formato `.pdf`
- Una versión digital de **todos** los archivos de código fuente, separados del informe, en un archivo comprimido (`.zip` o `.tar.gz`).

Los dos últimos deben enviarse a la dirección `tps.7541rw@gmail.com`, colocando como asunto:

```
TP2 - Padrón - Apellido
```

Se aclara que por código fuente se entiende todos los archivos `.h` y `.c`, el archivo `Makefile` para poder compilar, y todos los archivos adicionales que sean necesarios para ejecutar el programa. No deben entregarse nunca archivos `.o` u otros archivos compilados.

El informe impreso debe entregarse en clase. El plazo de entrega vence el **Lunes 11 de Noviembre de 2013**.